

# Dual-domain Hierarchical Classification of Phonetic Time Series

Hossein Hamooni  
Department of Computer Science  
University of New Mexico  
hamooni@cs.unm.edu

Abdullah Mueen  
Department of Computer Science  
University of New Mexico  
mueen@cs.unm.edu

**Abstract**—Phonemes are the smallest units of sound produced by a human being. Automatic classification of phonemes is a well-researched topic in linguistics due to its potential for robust speech recognition. With the recent advancement of phonetic segmentation algorithms, it is now possible to generate datasets of millions of phonemes automatically. Phoneme classification on such datasets is a challenging data mining task because of the large number of classes (over a hundred) and complexities of the existing methods.

In this paper, we introduce the phoneme classification problem as a data mining task. We propose a dual-domain (time and frequency) hierarchical classification algorithm. Our method uses a Dynamic Time Warping (DTW) based classifier in the top layers and time-frequency features in the lower layer. We cross-validate our method on phonemes from three online dictionaries and achieved up to 35% improvement in classification compared to existing techniques. We provide case studies on classifying accented phonemes and speaker invariant phoneme classification.

## I. INTRODUCTION

Phonemes are the smallest units of intelligible sound and phonetic spelling is the sequence of phonemes that a word comprises. For example, the word `boss` has two phonetic spellings for British (`/bɒs/`) and American (`/bɑːs/`) accents. In Figure 1, two versions of `boss` are shown with the phonemes labeled.

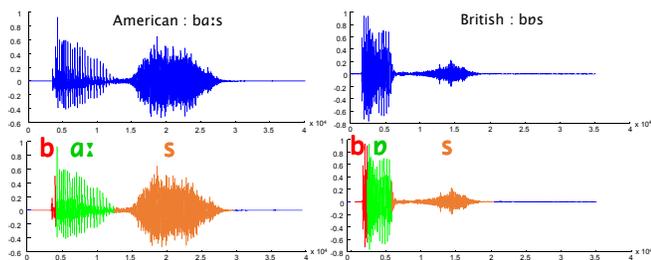


Fig. 1: Two waveforms of the word `boss` pronounced by American and British accented speakers. The American accent has a prolonged vowel, while the British accent does not. The British accent places less stress on the ending “s.” The perfect segmentation of the phonemes produced by Forced Aligner [25] is shown in color.

In this paper, we consider the problem of automatic phoneme classification that can be used for robust speech recognition, accent/dialect detection, speech quality scoring,

etc. Phoneme classification is inherently complex for two reasons. First, the number of possible phonemes is at least 107, based on the international phonetic alphabet (IPA). Therefore, this problem is a many class classification task for time series data. Second, phonemes suffer from variability in speakers, dialects, accents, noise in the environment, and errors in automatic segmentation.

There have been a plethora of works on phoneme classification in the signal processing and linguistics communities. Most of the works over the last twenty years are based on the classic TIMIT [11] dataset using statistical machine learning techniques. TIMIT is specifically designed for speaker-invariant phoneme classification. However, to build a robust phoneme classifier that can work in public environments with all kinds of variations, the classifier needs to learn from heterogeneous sources of data covering a large number of people, languages, age groups, etc. Therefore, we have taken a data-driven approach, which illustrates the phenomenon where a large amount of data can solve this complex problems with a simple and intuitive algorithm.

In this work, we have created a dataset of 370,000 phonemes automatically segmented from three online dictionaries covering the entire corpus of English words. We use a hierarchy of nearest neighbor classifiers using time and frequency domain features. We use a Dynamic Time Warping (DTW) based classifier in the top levels and frequency features in the lower layer. We adopt recent optimization techniques of time series similarity search for widely varying phoneme lengths. Our method performs 35% more accurately than other hierarchical classification techniques. We show case studies on the applicability of our classifier for accented phoneme recognition and speaker-invariant phoneme recognition.

The rest of this paper is organized to provide some background on phoneme classification in the beginning. We describe our data cleaning process to set more context in Section III. We move to describing our algorithmic techniques in Section IV and V. The last two Sections VI and VII discuss the experimental findings and case studies on related problems.

## II. BACKGROUND AND RELATED WORK

The number of phonemes in a language varies with the dialects of that language. The complete set of phonemes and their hierarchical organization have been made by the International Phonetic Association. There are 107 letters, 52

diacritics, and four prosodic marks in the International Phonetic Alphabet (IPA) covering various languages [2]. There exists an extension of IPA for speech pathologists that contains even more phonemes.

In this paper, we focus on English phonemes. English is spoken by more than a billion people and is therefore the most variable language. We focus on a standard set [13] of 39 phonemes in American English; 22 consonants and 17 vowels. These phonemes can be organized in a taxonomy as shown in Figure 2. Articulation of phonemes vary based on context, person, age, health condition, etc. For example, the phoneme /L/ can be pronounced by folding the tongue inside and also by extending the tongue out.

Phoneme segmentation, classification, and recognition are the three main tasks for automated phonotactic processing. Segmentation finds the phoneme boundaries inside a speech sequence. Classification identifies each individual phoneme, and recognition decodes the sequence of phonemes in the presence of segmentation and classification errors. These three tasks are sequential and interdependent. A great classifier that works only on human segmented phonemes should not be the goal to build applications using this pipeline. Similarly, we should not train recognizers that work on human classified phonemes only. To the best of our knowledge, our work is the first attempt to do phoneme classification on automatically segmented data.

Segmenting phonemes from speech audio is a challenging task. The primary reason is that the phonemes often convolve with each other and become unintelligible without context. The Forced Aligner tool aligns the speech to the known transcript and produces a segmentation that is as good as a human would do [25]. Automated segmentation allows us to create massive archives of phonemes and move to applications-based on phonemes such as language detector, accent detector, robust speech recognizer and so on.

Although phonemes are units of sounds, they are not used in recognition tasks to their full potential because of the variations and segmentation problems described above. Current speech recognizers work on fixed window MFCC features and ignore phoneme boundaries. In contrast, if a good phoneme classifier was present, the speech or phoneme recognizers would work on a discrete sequence of phonemes instead of raw signals and windowed features, which would make the recognizer more powerful and robust. A good example of a phoneme recognizer using the Viterbi algorithm can be found in [22].

Most of the existing works on phoneme classification are based on the manually labeled dataset from the Linguistic Data Consortium named TIMIT [11]. This dataset is specially made for phoneme classification, containing samples from more than 600 speakers. There is a long chain of works on the TIMIT dataset that use a variety of techniques and report classification performance on the standard test set in the corpus. Carla *et al.* [6] have presented a nice survey on all the existing works on TIMIT dataset. A pick on the algorithms can be Neural network [23], Deep Belief network [17], SVM [8][20],

Boltzmann machine [14], HMM [13], etc. The best accuracy reported on the TIMIT dataset so far is 79.8% [17].

Until now, there was no attempt to cross-validate these methods on other datasets. We identify this as a loop-hole because we do not know if these methods are overfitted towards the TIMIT dataset. Our experiments show that there exists tremendous source bias which produces high accuracy when training and test data are both from the same source (see Experiments section). When a new test data is tested on these models, the accuracy drops significantly. Source bias is not desirable in publicly deployable applications. We are the first to report accuracies across several sources and our proposed data-driven method shows less bias across the sources.

### III. DATA COLLECTION AND CLEANING

We have collected English words from three sources with their audio files. These consist of 30,000 words from Google Translate, 3,000 words from oxforddictionaries.com and 45000 words from the Merriam-Webster online dictionary. Each of these sources have different features. Audio files collected from Google translate, Oxford, and Merriam-Webster dictionaries are recorded at 22050, 44100 and 11025 samples per second respectively. All of them have male and female speakers in different ratios. The Oxford dictionary includes British and American accent pronunciation for each word. The variation among the sources needs attention when cleaning the data.

After data collection, we segment waveforms of the words to generate phonemes. We use the Forced Aligner tool from the Penn Phonetics Laboratory [11], which segments a word based on its phonetic spelling in the dictionary. Figure 1 shows a segmentation of the word `boss` produced by the tool. Forced Aligner produces very good segmentation even in the presence of noise. However, the phoneme boundaries may include some portions of the adjacent phonemes (See Figure 3 for an example). Such alignment error will make the phonemes inherently noisy in addition to the variations due to words and speakers.

The segmentation process generated 370,000 phonemes in 39 classes covering the entire dictionary. The number of phonemes in each class is shown in Figure 4. The distribution of phonemes in our dataset is very close to the distribution of phonemes in the English language shown in [9]. The phonemes vary in lengths. Figure 4 shows the distribution of the lengths.

#### A. Silence Removal

The first and the last phonemes of a word typically have a silence period before and after them respectively (Figure 3). We process those phonemes to remove the silence parts. As we have only words in the dictionaries, we z-normalize the signals and trim the consecutive beginning and ending values that are less than a threshold 0.05. This process is sufficient for our purpose and produces very accurate results.

#### B. Generating MFCC

Mel-frequency cepstrum coefficients (MFCC) features have been used for phoneme classification for many years and our

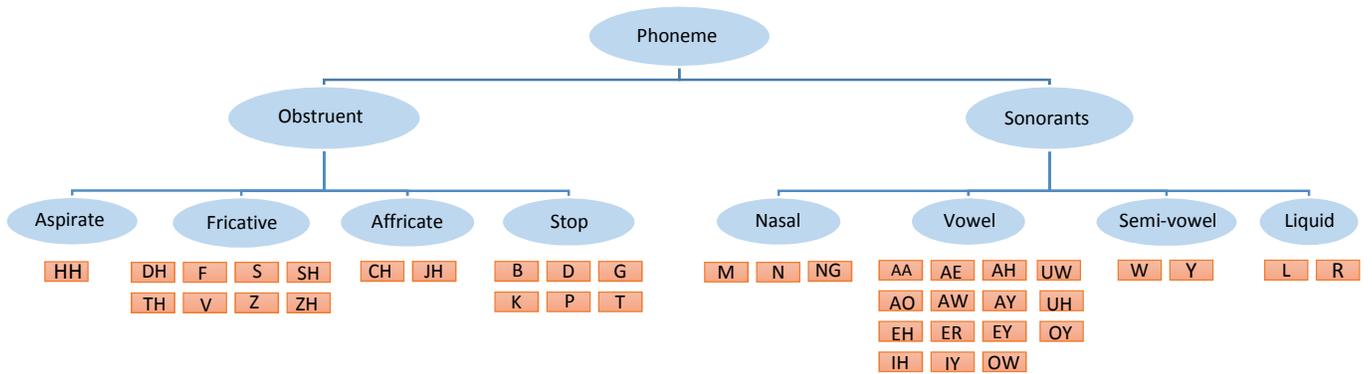


Fig. 2: The hierarchy of phonemes in English language. This is the most commonly used hierarchy for phoneme classification [8][13].

experiments also show their applicability in large scale classification. MFCC features convert the spectrum of an audio to the mel-scale that contains equally spaced pitches. We generate the MFCCs from the phonemes using standard 25 ms window slid by 10 ms. We use standard hamming window in the process. The coefficients are then averaged over all windows to produce the feature vectors of each phoneme. We normalize the individual coefficients across all the phonemes using z-normalization. We generate different number of coefficients for each phoneme. The number of features that are actually used for classification is 26 and justified in the experiments section.

### C. Resampling Waveforms

As we mentioned before, source bias is an unavoidable problem when multiple datasets are being used. Our data sources have different sampling frequencies. This variation can make the same phonemes with the same temporal durations have different lengths. To solve this problem, we do a resampling of the words to have all phoneme waveforms at 22050 sample per seconds. Resampling at the word level (not in the phoneme level) makes the same words roughly the same length. For example there are three different instances of the word `target` in the dataset. The lengths of the signals sampled at the original sampling rate are 12400, 20750 and 5100 points in Google, Oxford and Merriam-Webster datasets respectively. Comparing signals that vary exponentially in length is not desirable for similarity based techniques. We perform resampling and the lengths become 12400, 10375 and 10200 which are very close to one-another.

### D. Forced Aligning Waveforms

Forced Aligner tool segments the words and generates individual phonemes. The tool uses the phonetic transcription of a speech to extract the phonemes. For phonetic transcription, we use the CMU pronunciation dictionary [1] that contains approximately 125,000 words with their phonetic spellings. The algorithm of Forced Aligner finds the boundaries of the phonemes as well as any silent part in the speech. It is capable of segmenting long speech while we find the accuracies are

better for shorter speech samples. That is exactly the reason why we use dictionary of words instead of long ebooks.

As hinted before, boundaries of phonemes detected by Forced Aligner may not be perfect always. An example is in Figure 3 where the word `talk` is misaligned. The phoneme `/t/` is completely missed as silence and the vowel `/ɔ:/` is segmented into `/t/` and `/ɔ/`. The true number of such erroneous cases is not possible to determine on our dataset, however, the authors of [25] have mentioned in the original paper that Forced Aligner aligns *perfectly* and has very negligible difference with manual aligning.

Another limitation of Forced Aligner is the use of the pronunciation dictionary. Currently, we can segment 125,000 words which contains most of the words in English. However, there exist some proper nouns and compound words that are absent in the dictionary such as `Quakerism`, `Donatist` etc. This limitation is not limiting our classification algorithm as we can append the phonetic spellings of these words over time.

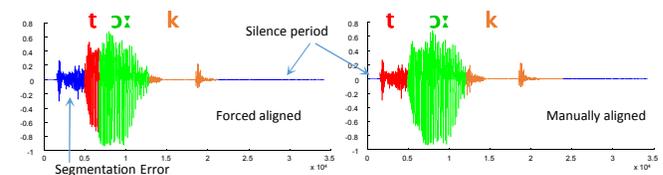


Fig. 3: Two segmentations of the word “talk” produced manually and by Forced Aligner. The later misaligned the phoneme `/t/`.

## IV. DTW BASED PHONEME CLASSIFICATION

We use  $K$  nearest neighbor algorithm (K-NN) for classification using Dynamic Time Warping (DTW) distance. There exist number of studies that find DTW as the most competitive distance measure for time series data [10]. For completeness, we provide the definition of DTW here. Let us define two signals  $x = x_1, x_2, \dots, x_n$  and  $y = y_1, y_2, \dots, y_m$  of length  $n$  and  $m$  where  $n > m$  without losing generality.

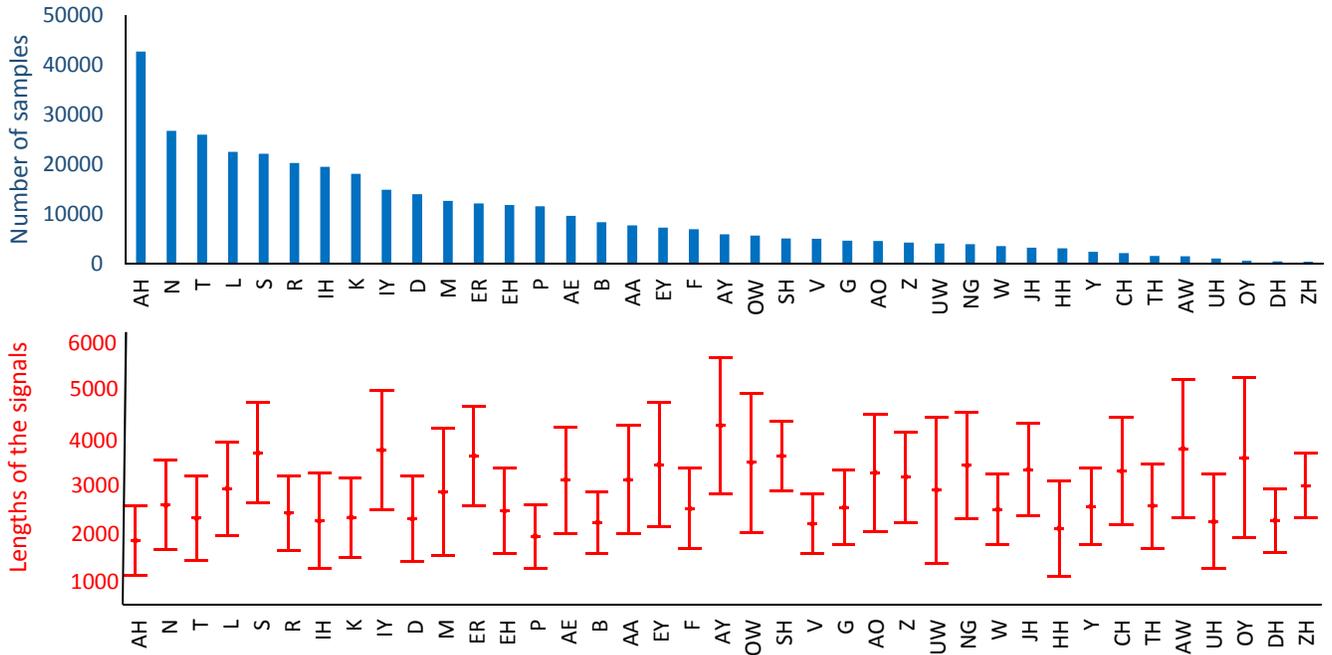


Fig. 4: The numbers of all the phonemes in our dataset in blue (top). The mean and standard deviation of the lengths of the signals after resampling in red (bottom).

$$DTW(x, y) = D(n, m)$$

$$D(i, j) = (x_i - y_j)^2 + \min \begin{cases} D(i-1, j) \\ D(i, j-1) \\ D(i-1, j-1) \end{cases}$$

$$D(0, 0) = 0, \forall_{ij} D(i, 0) = D(0, j) = \infty$$

We intentionally skip taking the square root of  $D(n, m)$  as it does not change the neighbors ordering while makes it efficient for speedup techniques. DTW is originally defined for signals of unequal lengths and computing DTW distance is quadratic in time complexity because it requires populating the entire  $n \times m$  matrix. Constrained DTW distance populates only a portion of the matrix around the diagonal. Typically constraints are expressed by a window size ( $w$ ) (readers can find details about DTW in any online resource and also in [12]). When two signals are quite different in lengths/durations, there are two approaches to compute constrained DTW distance. First, by resampling one of the signals to the size of the other and thus, making them equal in size. This approach works with arbitrary size of the window because  $D$  is a square matrix. Note that, constraining the warping path between  $[-w, w]$  of the diagonal of a rectangular matrix has the same effect of resampling/interpolating the shorter signal to the size of the longer one. The second approach is to impose a condition  $w \geq |x - m|$  on the window size and compute the matrix as defined above. This approach does not resample any of the signals. Both the approaches have quadratic time complexities.

There are dozens of papers that describe speeding up techniques for DTW based similarity search. Lower bounding

technique [12], early abandoning technique [18], hardware acceleration technique [21], summarization technique [16], anticipatory pruning [5], etc. All of these works use the resampling approach mentioned above and take benefit of the *LB\_Keogh* bound for efficiency and effectiveness in similarity search. For completeness, we describe *LB\_Keogh* here. *LB\_Keogh* generates two envelopes ( $U$  and  $L$ ) of a candidate and compares them with the query to produce the lower bound. *LB\_Keogh* takes linear time for computation and insignificant off-line preprocessing for the envelopes. Figure 6(left) shows a toy signal in blue and its envelopes in red for different window sizes. The formula to compute  $U$ ,  $L$  and *LB\_Keogh* where  $w$  is the constraint window is given below.

$$\forall i U_i = \max(x(i-w : i+w))$$

$$\forall i L_i = \min(x(i-w : i+w))$$

$$LB\_Keogh = \sum_{i=1}^n \begin{cases} (y_i - U_i)^2 & \text{if } y_i > U_i \\ (y_i - L_i)^2 & \text{if } y_i < L_i \\ 0 & \text{otherwise} \end{cases}$$

*LB\_Keogh* and many other recent optimization techniques for DTW based similarity search are only defined for equal-length signals after resampling and shown to be successful in many application areas. However, phonemes are variable in lengths because of the speaker variations. Moreover, errors from Forced Aligner make the lengths even more variable. Figure 4 shows the length distribution of the phonemes in our dataset which demonstrates a wide variance in each of the phonemes in English. To adopt the existing techniques

for fast similarity search, we *cannot* resample the signals to a fixed length. Resampling can have detrimental impact on time domain techniques such as computing DTW distance on phonemes. Figure 5(left) shows an example of resampling error for phonemes where a subsequence (green/bottom) of a larger phoneme (blue/middle) generates more distance than it does to some random phoneme (red/top). To get a better picture, we calculate 5000 DTW distances with and without resampling and create a scatter plot. Figure 5(right) circles the pairs of signals that could be near neighbors of each other under regular DTW but, are moved far when resampled to equal lengths.

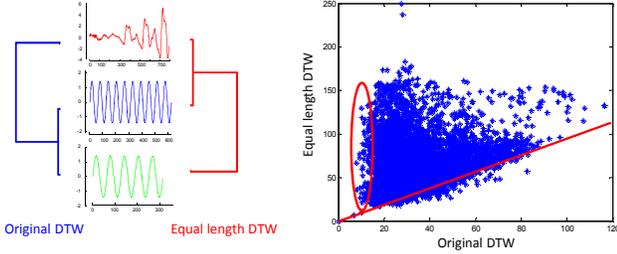


Fig. 5: (left) An example of erroneous association caused by DTW distances on resampled equi-length signals. (right) The scatter plot shows there exist numerous other cases of similar signals being measured as dissimilar by equi-length DTW.

The original constrained DTW distance for signals with unequal lengths is computationally expensive. We want to use lower bounding technique to speed up the search. Although commonly used bounds, *LB\_Kim*, *LB\_Yi* and *LB\_Keogh*, are defined for equi-length signals, we can adopt each of them by using the *prefix* of the longer signal. Surprisingly, such intuitive property of these bounds was not used previously probably because there was no such need where different lengths of signals are classified using DTW. For example, Figure 6(right) shows the prefixed *LB\_Keogh* or in short, *PLB\_Keogh* computation that ignores all the  $x_i, i = m + 1, \dots, n$ . Recall, the prefix *LB\_Keogh* is a bound only under the condition  $w \geq |n - m|$ . The proof follows the same reasoning as the original proof for *LB\_Keogh* [12] and is omitted here.

In addition to *PLB\_Keogh*, we can use the early abandoning DTW computation which abandons computation as soon as the minimum value of a row in the DTW matrix is larger than the minimum distance found so far. Since *PLB\_Keogh* is not symmetric, we can switch the role of the query and candidate to generate two bounds and use the larger. Note that a query can be longer than some candidates and shorter than others which does not prevent applying the techniques. However, we cannot apply the early abandoning z-normalization technique for phonemes because training phonemes are independent of each other as opposed to overlapping subsequences of a long signal. Table I shows a list of techniques from [18] and their applicability in phoneme classification.

As we have a set of techniques applicable to our problem,

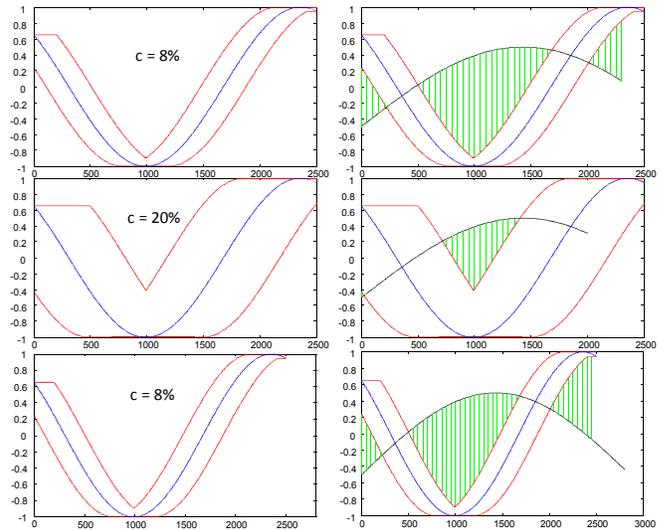


Fig. 6: (left) Envelopes  $U$  and  $L$  for the candidate (blue) sinusoid over different windows. (right) *LB\_Keogh* using the prefix of the longer signal. (top) The query shown in black is the shorter of the signals of length 2300. Note that, the difference in length between the blue and the black signals is equal to the window size 8% of the longer signal. (middle) the query in black is of length 2000 where window size is 20% of the longer. (bottom) the query in black is longer than the blue candidate and only its first 2500 samples are used for the bound.

TABLE I: Techniques Applicable for Phonemes

Techniques	Applicability
Squared distance	Yes
Lower bounding	Yes
Early Abandoning of DTW	Yes
Exploiting multicore	Yes
Early abandoning z-normalization	No
Reordering early abandoning	Yes
Reversing roles of query and candidates	Yes

there is one piece of puzzle that needs attention. The window size  $w$  is a critical parameter. Previously, it was set to a fixed value such as 30 samples or 30% of the fixed length. We don't have the luxury anymore as there is the condition  $w \geq |n - m|$  which depends on the pairs of signal. There are two options. First, we could trivially set  $w = |n - m|$ . This is both inefficient and unnecessary because phonemes with significant difference in length (e.g. 3000 vs. 500) are unlikely to be neighbor of each other while their DTW distance computation populates almost the entire matrix. Second, we can set  $w = c * \max(n, m)$  and ignore the pairs for which  $w < |n - m|$ . We use  $c = 30\%$  in our experiments and justify this choice in the experiments section.

DTW distances computed from various pairs are not directly usable in K-NN algorithm. Because shorter candidates will have a bias. For example, a signal of length 1000 will be more biased to match with signals of length 700 to 1000 than

---

**Algorithm 1** *PhonemeClassification(Train, Test, c, K)*

---

**Ensure:** Return the labels for the signals of the Test

```
1: Preprocessing
2: for  $x \leftarrow Train_i$ ,  $i = 1, 2, \dots, |Train|$  do
3:   Calculate  $U_x$  and  $L_x$  using  $w_x \leftarrow \lfloor c * |x| \rfloor$ 
4: end for
5:
6: Testing
7:  $d_K \leftarrow \infty$ 
8: for  $y \leftarrow Test_j$ ,  $j = 1, 2, \dots, |Test|$  do
9:   Calculate  $U_y$  and  $L_y$  using  $w_y \leftarrow \lfloor c * |y| \rfloor$ 
10:  for  $x \leftarrow Train_i$ ,  $i = 1, 2, \dots, |Train|$  do
11:    if  $|x| \geq |y|$  and  $w_x \geq (|x| - |y|)$  then
12:       $LB \leftarrow PLB\_Keogh(x, y, U_x, L_x, w_x)$ 
13:       $w \leftarrow w_x$ 
14:    else if  $|y| \geq |x|$  and  $w_y \geq (|y| - |x|)$  then
15:       $LB \leftarrow PLB\_Keogh(y, x, U_y, L_y, w_y)$ 
16:       $w \leftarrow w_y$ 
17:    end if
18:    if  $LB < d_K$  then
19:       $d \leftarrow NormalizedDTW(x, y, w, d_K)$ 
20:      if  $d < d_K$  then
21:         $d_K \leftarrow d$ , Save the neighbor  $x$ 
22:      end if
23:    end if
24:  end for
25:  Use the K-NNs to find the phoneme class
26: end for
```

---

to signals of length 1000 to 1300 for  $c = 30\%$ . It will not be fair if we pick the nearest neighbor without a measure for this. Some previous works [15] show a simple length normalization technique for Euclidean distance. Similarly, we normalize the DTW distances by dividing it with the lengths of the shorter signals of the pairs,  $NormalizedDTW(x, y) = DTW(x, y) / \min(n, m)$ . We show the classification process in the Algorithm 1. The value of  $K$  is set to 7 for all our experiments and is justified in the experiments section.

## V. DUAL DOMAIN PHONEME CLASSIFICATION

DTW based classifier uses the temporal similarity of the signals to classify the phonemes. Such use of temporal similarity for phoneme classification is unprecedented. All of the previous works perform classification on a set of features dominated mostly by MFCC features and achieve competitive accuracy. We investigate if combining both time and frequency domain techniques along with similarity based classification gives us better results than any of them individually.

To use multiple domains, we resort to the hierarchy of the phonemes shown in Figure 2. We classify the first two layers by our DTW based K-NN classifier and the final layer is classified using K-NN algorithm with MFCC features. The top two layers construct an eight class problem and the bottom layer constructs as large as 15 class problem for vowels. Training a classifier using the phoneme hierarchy has been

done previously [8] using MFCC features. Using multiple classification techniques at different hierarchy has also been done [7][23]. We introduce the dual-domain hierarchical classification for phonetic time series and use K-NN for the first time on phonemes.

The intuition behind using both time domain and frequency domain features is that both are important for phonemes. For example, the same phoneme can be pronounced with different durations. For instance, `foot` and `food` have short and long “o” sounds which vary in temporal duration. In contrast, vowels and consonants vary in their frequencies. The intuition behind using DTW based technique ahead of MFCC features is a bit counter intuitive specially if we consider the potential gain in speed of a classifier that runs MFCC in the top layer and DTW in the bottom. Since MFCC features are compressing the information into small number of coefficients, the loss in accuracy becomes detrimental. Yet, MFCC features can capture the tiny differences between classes that are washed away when DTW is computed on the raw signal. The experimental validation of this intuition is provided in the experiments section.

To describe the idea more clearly, we show two examples in Figure 7. We can generate many of such examples where hierarchical method can classify a phoneme correctly while none of the MFCC or DTW based classification algorithms can do it individually. Lets take the “SH” example. MFCC features can match the query with a very dissimilar waveform (CH) while DTW can find closely similar signal that sufficiently identifies the general class (Fricative) of the phoneme, although the specific class “S” is wrong. However, MFCC works well finding the right label within the general class and correctly lands at the “SH” node. Note that the source word of the unknown sample and the source words of the matched phonemes are completely different.

## VI. EXPERIMENTS

We perform an extensive set of experiments to demonstrate the effectiveness of our classification techniques. We claim 100% reproducibility of our work. We have all of our data, code, results, slides etc. uploaded to [4] and it’s publicly available. Unlike others, we have also uploaded the massive distance matrix of size  $3000 \times 370000$  for interested researchers.

As mentioned earlier, we have dictionary words from three sources. We have 370,000 phonemes in total. For simplicity, we use **GG** for the phonemes from Google Translate, **OO** for Oxford and **WW** for Merriam-Webster. GG has approximately 200,000 phonemes, OO has 30,000 and WW has 140,000 phonemes. For most of our experiments we combine all three sets in a large pool of phonemes unless otherwise specified.

Our method is implemented in Matlab for data input and output and the core DTW function and lower bounds are implemented in C and compiled using mex compiler to integrate with Matlab. All the experiments have been done on a desktop machine running Ubuntu 12.04 with Intel Core i7 processor and 32 GB of memory. All reported accuracies are calculated

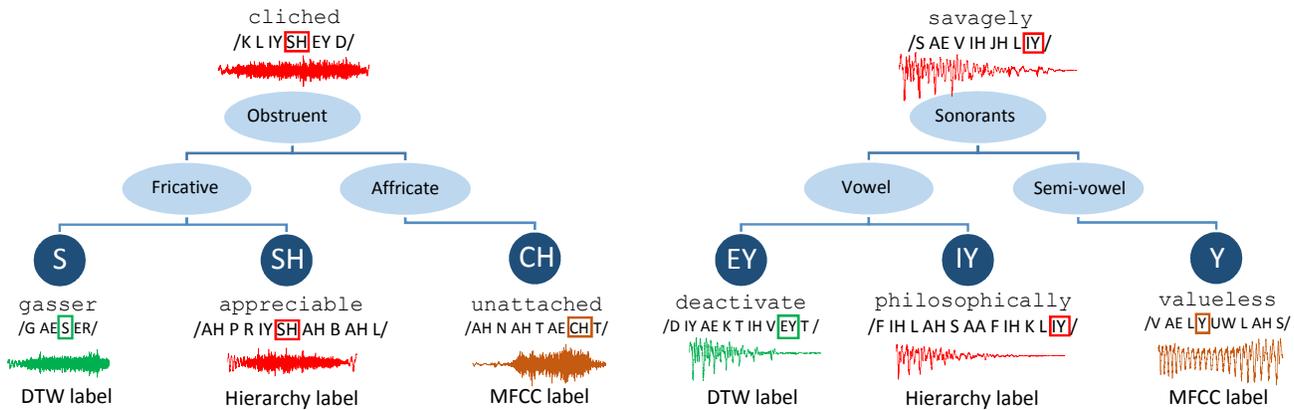


Fig. 7: Two examples. Each tree shows the unknown phoneme, its waveform and the source word at the root. The three leaf nodes of a tree show the three classes the algorithms produce. In each of the leaf nodes, the nearest neighbor phoneme, its waveform and the source word are shown.

using 10-fold cross validation using 100 random phonemes as test set at each fold. The number of neighbors and the size of the window are 7 and 30% respectively.

a) *Competitor Algorithms:* There exists a varieties of algorithms that have been reported on the TIMIT dataset. The only known algorithm for phoneme classification that uses the phoneme hierarchy is presented in [8]. It is an online algorithm that reads the phonemes once and update the model iteratively after each phoneme is processed. The algorithm can be used in batch mode and we have implemented the algorithm in Matlab to test on our dataset. The code is available in [4]. We use the abbreviation of the title of the paper, OAHPC, as the name of the algorithm in the subsequent sections.

The PhoneRec [3] is a complete tool that takes in a speech file and outputs the sequence of phonemes. However, we find that the tool is severely erroneous on our datasets when we use the pre-trained models. For example, the word *egocentric* is converted to the phoneme sequence /iy d ih l er s ih m t ih g k t/. Another example, the word *greasy* is converted to the phoneme sequence /hh uw iy z iy t/. We use a set of 200 phonemes and the accuracy is less than 10%. We don't think it would be a fare comparison as we assume the phonemes are already segmented while PhoneRec tool combines all the three steps (segmentation, classification and recognition) in a single tool.

We have made effort to reproduce the best reported algorithm for TIMIT dataset [17]. We were unable to reproduce from the paper and there was no code and documentation available from the authors. Recall TIMIT [11] is a manually labeled data and it costs at least \$250 to obtain license for the data.

b) *Classification in the Top Two Layers:* Our first experiment is to find the best classifier for the top two layers which is essentially an eight class problem (See Figure 2). We use DTW based K-NN classifier and MFCC based K-NN classifier using Euclidean distance. We also test OAHPC on these layers. We experiment on three test sets separately. In each experiment,

we start with a training set of 10,000 phonemes and add 20,000 phonemes to it iteratively. The results are shown in Figure 8. The accuracy increases as we add more and more data. We observe DTW is better in classifying the top layers than MFCC as described before. The performance of OAHPC is not promising to use in these two layers.

c) *Hierarchical Phoneme Classification:* We experiment to measure accuracy in classifying 39 phonemes at the leaves of the hierarchy. We perform the experiments using five different methods. We use DTW and MFCC individually for K-NN algorithms. We test dual-domain classifiers by using DTW first, MFCC next and vice versa. We also test the OAHPC algorithm. Incremental accuracies are reported on the whole of 370,000 phonemes. The results are shown in Figure 9. We find DTW+MFCC as a better combination than MFCC+DTW. K-NN classifiers, in general, work better than OAHPC.

The differences in accuracies for the three test sets are significant because of the unbalanced contributions from each of the sources. The most important point in this experiment is the trends of the curves. As we increase the number of phonemes in the training set, the accuracies are increasing. Although the rate is diminishing, this is promising because it can be the basis of applications that index speech samples of all the people of the earth.

d) *Parameter Sensitivity:* Our hierarchical classification method has exactly two parameters, the number of neighbors (K) that vote for the class and the window size (c). We perform experiments to find the most suitable value for each of these parameters. We keep one of them fixed to  $K = 15$  or  $c = 35\%$  and vary the other over a range. We measure the mean accuracy over a random test set. The results are shown in Figure 10. We tested on randomly chosen 60,000 phonemes from our dataset.

We observe a clear peak at  $K = 7$  and therefore, we use 7-NN methods in all our experiments. While varying  $c$ , we observe a small increase upto 30% and more than that, there

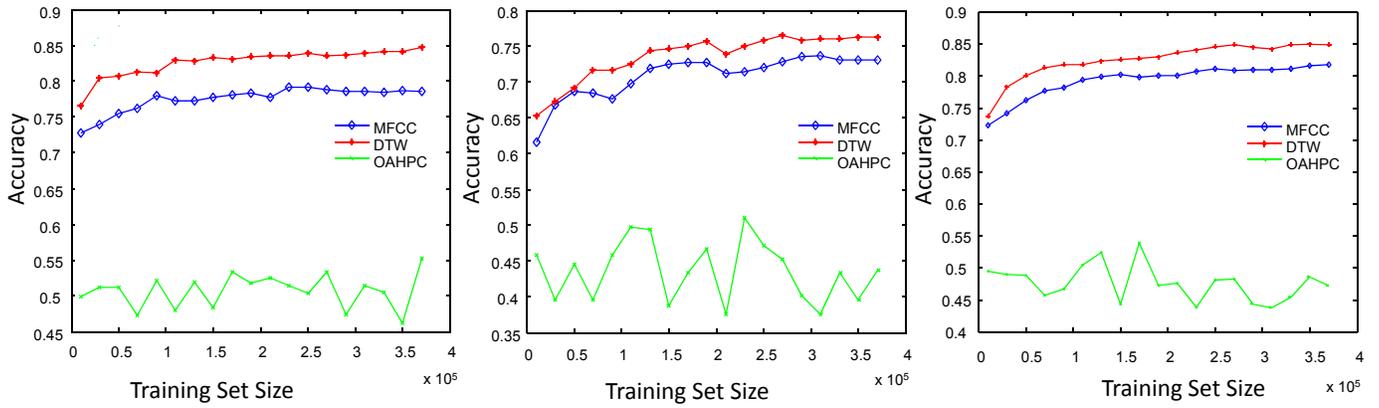


Fig. 8: Incremental accuracy of the algorithms in the top two layers of the hierarchy on three test sets. (left) **GG** (middle) **OO** and (right) **WW**.

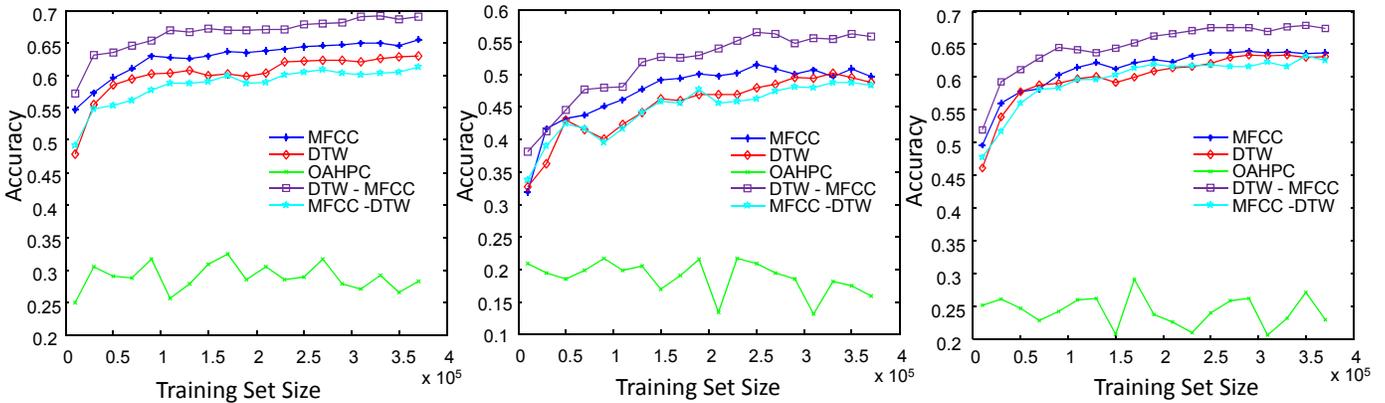


Fig. 9: Incremental accuracy of the algorithms in whole of the hierarchy on three test sets. (left) **GG** (middle) **OO** and (right) **WW**.

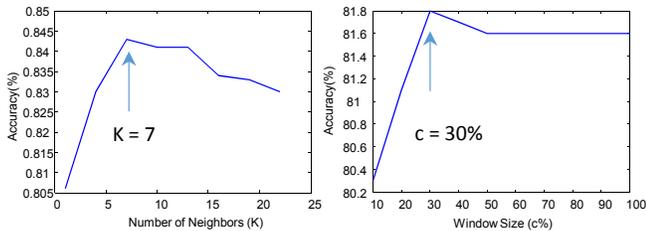


Fig. 10: (left) Accuracies for different  $K$  at the top two layers. (right) Accuracies for different window sizes ( $c$ ) at the top two layers.

is no impact on accuracy. Therefore, we use 30% as window size for all of our experiments.

*e) Sliding DTW Based Phoneme Recognition:* In this work, we have tested another algorithm for phoneme classification which we describe very briefly here. The motivation of the algorithm is to avoid the segmentation process completely by using the Spring [19] method of finding the matching

location of a pattern inside a long signal under warping distance. The algorithm, in a nutshell, searches for the  $K$  words where an unknown phoneme matches best at any location. The matching locations inside the  $K$  words vote to produce a phoneme for the unknown sample. For example, for  $K = 3$ , an unknown phoneme may match with *cross*, *saturn* and *cell* at their starting locations. Since all of these locations vote for “s”, the algorithm produces “s” as the class label.

We test the above algorithm against our proposed hierarchical algorithm. The results show a superiority of the hierarchical method. The biggest drawback of the above algorithm is that a phoneme can match in between two phonemes in a word. This added confusion hurts the accuracy significantly and thus, it supports the use of Forced Aligner for segmentation before doing the phoneme classification.

*f) Scalability:* We adopt all the techniques from [18] and perform scalability experiment on 200,000 phonemes. We randomly pick 20 phonemes and run the similarity search algorithm over 200,000 phonemes. The window size is 30%. We achieve around 2 times speedup over the original implementation without any optimization. Figure 11 shows the speedup

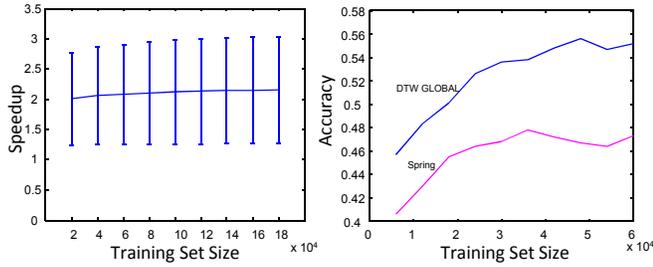


Fig. 11: (left) Speedups achieved using the techniques in the Table I (right) Comparison between Spring [19] based classifier with global DTW based K-NN classifier. The former requires no segmentation and the later uses Forced Aligner.

curve. We achieve the best speedup from early abandoning technique. All the other techniques contribute equally. The reason why we cannot achieve the dramatic speedup as [18] is explainable. Since we cannot resample, the window size has to be larger than the length difference. As shown in Figure 4, the standard deviations of lengths are quite large, therefore, most of the DTW computations are done for a large sized window. A large window essentially converts *PLB\_Keogh* to *LB\_Yi* [24] which is a very weak bound. Figure 12 shows the visual proof of *PLB\_Keogh* working for 40,000 pairs of signals of different lengths where there is no counter example found.

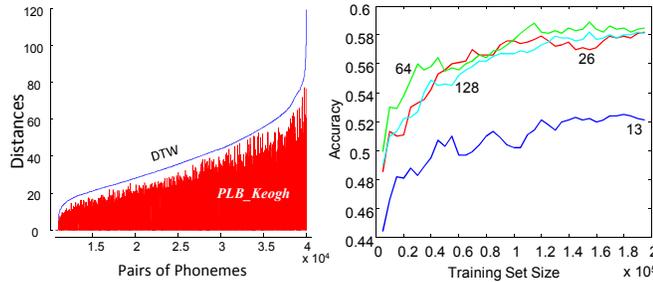


Fig. 12: (left) Visual proof for *PLB\_Keogh* bound. (right) Sensitivity of the number of MFCC coefficients used for the classifiers.

g) *Number of MFCC Coefficients*: We experiment to find the best number of MFCC coefficients. We run the K-NN classifier for all the phonemes in the GG dataset using first 13, 26, 64 and 128 MFCC coefficients. The results are shown in Figure 12(right). We see insignificant difference among 26, 64 and 128. Although most of the previous work use 13 MFCC coefficients we see 13 coefficients perform less accurate. Based on the results we use 26 features for all of our experiments.

## VII. CASE STUDIES

### A. Accented Phoneme Classification

Accent is a type of variation in the spoken form of a language. In this case study, we investigate if our method can classify accented phonemes. We have collected approximately

30,000 phonemes from the oxforddictionaries.com with close to 50-50 distribution of British and American pronunciations. Our first experiment is targeted to see the difference in accuracies when we introduce accented phonemes in the test set. Figure 13 shows the incremental accuracies of the two test sets; British accented phonemes and American accented phonemes. We see a reasonable drop (6%) in performance for British accented phonemes. The reason is that all the phonemes in GG and WW datasets are in American accent and therefore, the training data is biased with around 95% American accented phonemes. With such a massive bias in the training data, 50% accuracy in a 39-class problem is a considerable number.

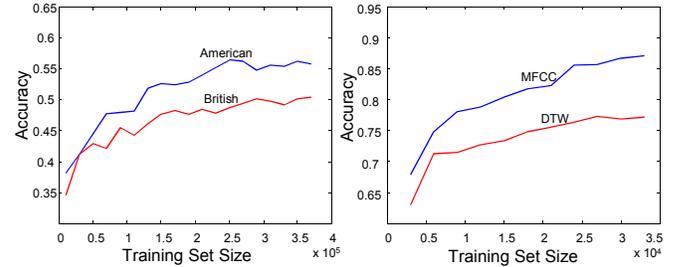


Fig. 13: (left) Accented phoneme classification accuracies using the whole dataset. (right) Accent detection using K-NN classifiers.

We further investigate if our classification strategy can detect the accent based on the phonemes only. It is a hard task even for humans as phonemes are only few milliseconds long and sometimes it is hard to even identify the phoneme itself. Since this is a 2-class problem, we cannot use the hierarchical method. We use the DTW and MFCC based K-NN algorithms for this task. We see MFCC performing extremely well.

### B. Speaker Invariant Phoneme Classification

In previous experiments, training set contains phoneme from all the three different datasets. Here we want to measure classification accuracy of our method when test set and training set are from two different sources e.g. test set contains phoneme of OO dataset and train set contains phonemes of WW and GG datasets. This ensures that there is no common speaker in the training and testing set. As there is source bias in the datasets, we expect less accuracy when performing speaker invariant detection of phonemes.

Table II and III show our results. We see that dual-domain hierarchical classification gains significantly more accuracy over OAHPC. Most importantly, dual-domain classifier shows better accuracies when tested against both of the remaining datasets than accuracies of those datasets individually. OAHPC does not show this behavior. This is an evidence that even though we have around 40% accuracy for speaker invariant phoneme detection, we can expect better performance as we increase data from other sources.

TABLE II: Speaker Invariant Accuracy using Dual-domain Hierarchy

Testset	GG	OO	WW	Except Testset
GG	–	38.5	45.7	<b>46.8</b>
OO	30.5	–	37.5	<b>39.3</b>
WW	43	38.5	–	<b>47.0</b>

TABLE III: Speaker Invariant Accuracy using OAHPC

Testset	GG	OO	WW	Except Testset
GG	–	14.2	19.0	<b>19.9</b>
OO	<b>20.8</b>	–	13	15.8
WW	<b>24.9</b>	15.8	–	18.6

### VIII. CONCLUSION

In this paper, we present a dual-domain hierarchical classification technique for phonetic time series data. This technique has a significant application in classifying English phonemes and is the first similarity based technique used for such problem. We use a novel dataset of 370,000 phonemes generated from three online dictionaries. Our experiments show that the data-driven phoneme classification method has promising capabilities when training set grows with samples from heterogeneous sources. This work is just an introduction of the phoneme detection problem to the data mining community. In future, we will work on phoneme recognition using our classifier as well as introducing other types of variability such as contextual and behavioral changes.

### REFERENCES

[1] The cmu pronouncing dictionary. <http://www.speech.cs.cmu.edu/cgi-bin/cmudict>.

[2] International phonetic alphabet. [https://en.wikipedia.org/wiki/International\\_Phonetic\\_Alphabet](https://en.wikipedia.org/wiki/International_Phonetic_Alphabet).

[3] Phoneme recognizer based on long temporal context. <http://speech.fit.vutbr.cz/software/phoneme-recognizer-based-long-temporal-context>.

[4] Repository for supporting materials. [http://cs.unm.edu/~hamooni/papers/Dual\\_2014/index.html](http://cs.unm.edu/~hamooni/papers/Dual_2014/index.html).

[5] I. Assent, M. Wichterich, R. Krieger, H. Kremer, and T. Seidl. Anticipatory dtw for efficient similarity search in time series databases. *Proc. VLDB Endow.*, 2(1):826–837, 2009.

[6] L. Carla and P. Fernando. Phoneme recognition on the timit database. *Speech Technologies*, 2011.

[7] N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Hierarchical classification: Combining bayes with svm. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 177–184, 2006.

[8] O. Dekel, J. Keshet, and Y. Singer. An online algorithm for hierarchical phoneme classification. In *Proceedings of the First International Conference on Machine Learning for Multimodal Interaction*, MLMI'04, pages 146–158, 2005.

[9] E. DEWEY, Godfrey. *Relative Frequency of English Spellings*, pages 69–84. 1970.

[10] H. Ding, G. Trajcevski, X. Wang, and E. Keogh. Querying and mining of time series data: Experimental comparison of representations and distance measures. In *In Proc of the 34 th VLDB*, pages 1542–1552, 2008.

[11] J. Garofolo. Timit acoustic-phonetic continuous speech corpus ldc93s1. *Web Download. Philadelphia: Linguistic Data Consortium*, 1993.

[12] E. Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 406–417, 2002.

[13] K.-F. Lee and H.-W. Hon. Speaker-independent phone recognition using hidden markov models. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 37(11):1641–1648, 1989.

[14] A. Mohamed and G. Hinton. Phone recognition using restricted boltzmann machines. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 4354–4357, 2010.

[15] A. Mueen. Enumeration of time series motifs of all lengths. In *ICDM*, pages 547–556, 2013.

[16] F. Petitjean, A. Ketterlin, and P. Ganarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, 44(3):678 – 693, 2011.

[17] A. rahman Mohamed, G. E. Dahl, and G. E. Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech & Language Processing*, 20(1):14–22, 2012.

[18] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. *KDD '12*, pages 262–270, 2012.

[19] Y. Sakurai, C. Faloutsos, and M. Yamamuro. Stream monitoring under the time warping distance. *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, 0:1046–1055, 2007.

[20] J. Salomon. Support vector machines for phoneme classification. *Master of Science, School of Artificial Intelligence, Division of Informatics, University of Edinburgh*, 2001.

[21] D. Sart, A. Mueen, W. Najjar, V. Niennattrakul, and E. J. Keogh. Accelerating dynamic time warping subsequence search with gpus and fpgas. In *ICDM*, pages 1001–1006, 2010.

[22] P. Schwarz. Phoneme recognition based on long temporal context. In *PhD Thesis, Brno University of Technology*, 2009.

[23] P. Schwarz, P. Matejka, and J. Cernocky. Hierarchical structures of neural networks for phoneme recognition. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, 2006.

[24] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings of the Fourteenth International Conference on Data Engineering, Orlando, Florida, USA, February 23-27, 1998*, pages 201–208, 1998.

[25] J. Yuan and M. Liberman. Speaker identification on the scotus corpus. In *In Proceedings of Acoustics 2008*, 2008.