Knowl Inf Syst (2016) 48:253–275
DOI 10.1007/s10115-015-0885-9

REGULAR PAPER

Phoneme sequence recognition via DTW-based classification

Hossein Hamooni[1] · Abdullah Mueen[1] · Amy Neel[2]

Received: 25 November 2014 / Revised: 20 August 2015 / Accepted: 5 October 2015 /
Published online: 19 October 2015

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 with existing techniques. We further modify our vowel classifier by adopting DTW distance over time–frequency coefficients and gain an additional 3 % improvement. We provide case studies on classifying accented phonemes and speaker-invariant phoneme classification. Finally, we show a demonstration of how phoneme classification can be used to recognize speech.

Keywords Phoneme classification · DTW-based classification · Phonetic time series · Big data · Sequence recognition

✉ Hossein Hamooni
hoseinhamooni@gmail.com; hamooni@cs.unm.edu

Abdullah Mueen
mueen@cs.unm.edu

Amy Neel
atneel@unm.edu

1 Department of Computer Science, University of New Mexico, Albuquerque, NM, USA

2 Department of Speech and Hearing Sciences, University of New Mexico, Albuquerque, NM, USA

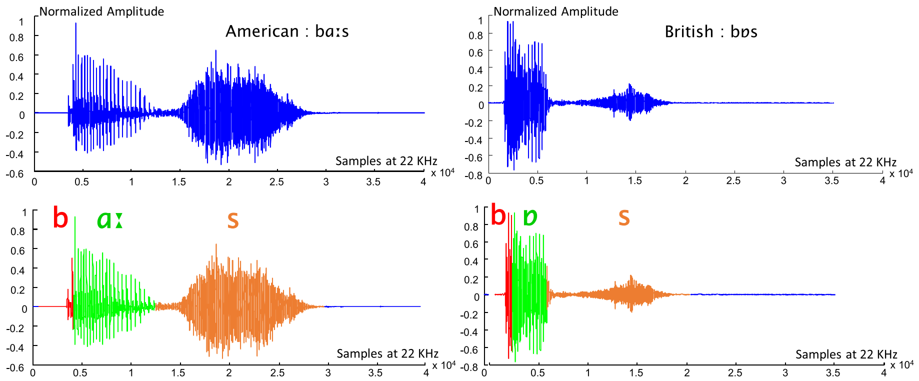


Fig. 1 Two waveforms of the word `boss` pronounced by American- and British-accented speakers. The American accent has a prolonged vowel, while the British accent does not. The British accent places less stress on the ending "s." The perfect segmentation of the phonemes produced by Forced Aligner [1]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

1 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 Fig. 1, two versions of `boss` are shown with the phonemes labeled.

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. In addition to the phoneme classification method described in the original paper [2], we introduce a word detection method which works on the sequence of classified phonemes and is able to correct some phoneme classification errors. We also introduce a method to improve accuracy of classification on the most difficult subset of phonemes, called *vowels*.

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 20 years are based on the classic TIMIT [3] 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 problem with a simple and intuitive algorithm.

In this work, we have created a dataset of 376,509 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

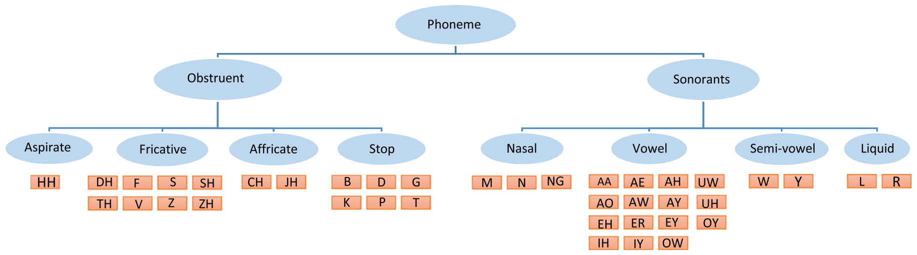


Fig. 2 Hierarchy of phonemes in English language. This is the most commonly used hierarchy for phoneme classification [5,6]

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 Sect. 3. We move to describing our algorithmic techniques in Sects. 4 and 5. The last two Sects. 7 and 8 discuss the experimental findings and case studies on related problems.

2 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 IPA covering various languages [4]. 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 [5] of 39 phonemes in American English: 22 consonants and 17 vowels. These phonemes can be organized in a taxonomy as shown in Fig. 2. Articulation of phonemes varies 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 [1]. Automated segmentation allows us to create massive archives of phonemes and move to applications based on phonemes such as language detector, accent detector and robust speech recognizer.

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 (mel-frequency cepstrum coefficients) 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.

Most of the existing works on phoneme classification are based on the manually labeled dataset from the Linguistic Data Consortium named TIMIT [3]. 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. [7] have presented a nice survey on all the existing works on TIMIT dataset. A pick on the algorithms can be neural network [8], deep belief network [9], SVM [6,10], Boltzmann machine [11], HMM [5], etc. The best accuracy reported on the TIMIT dataset so far is 79.8 % [9].

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 whether these methods are overfitted toward 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 new test data are 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.

3 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 45,000 words from the Merriam-Webster online dictionary. Each of these sources has different features. Audio files collected from Google translate, Oxford and Merriam-Webster dictionaries are recorded at 22,050, 44,100 and 11,025 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 [3], 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 Fig. 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 376,509 phonemes in 39 classes covering the entire dictionary. The number of phonemes in each class is shown in Fig. 4. The distribution of phonemes in our dataset is very close to the distribution of phonemes in the English language shown in [12]. The phonemes vary in lengths. Figure 4 shows the distribution of the lengths.

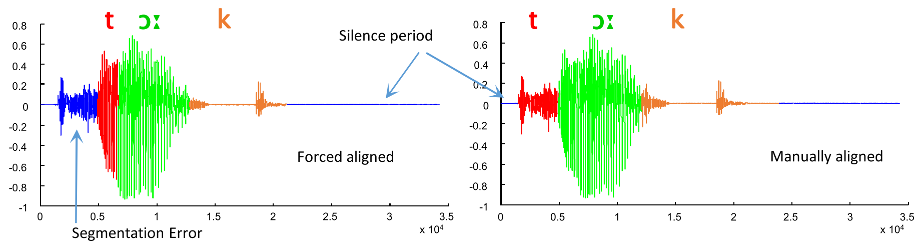


Fig. 3 Two segmentations of the word “talk” produced manually and by Forced Aligner. The latter misaligned the phoneme /T/

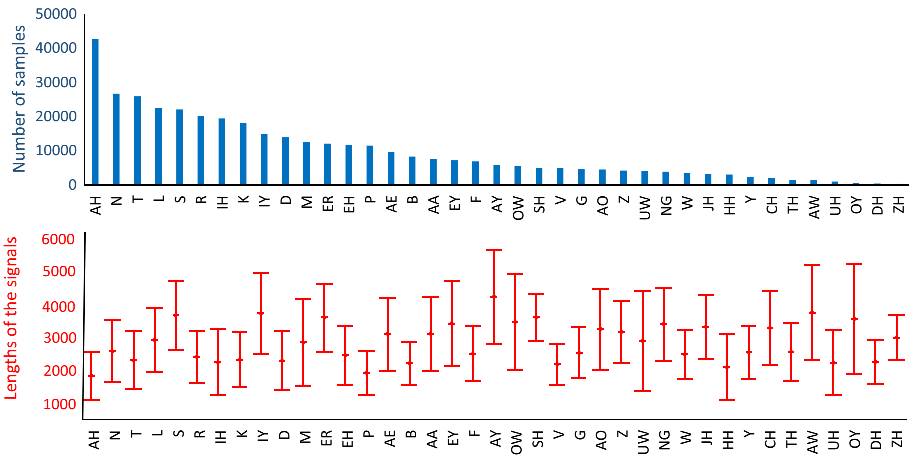


Fig. 4 Numbers of all the phonemes in our dataset in *blue* (*top*). The mean and standard deviation of the lengths of the signals after resampling in *red* (*bottom*) (color figure online)

3.1 Silence removal

The first and the last phonemes of a word typically have a silence period before and after them, respectively (Fig. 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.

3.2 Generating MFCC

Mel-frequency cepstrum coefficients (MFCC) features have been used for phoneme classification for many years (see [7] for a complete list), and our 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 in increasing frequency order. We generate the MFCCs from the phonemes using standard 25 ms window slid by 10 ms (see [13] for code). We use standard Hamming window in the process. For example, an audio sample of 1 s produces 100 windows, and each window generates a given number of MFCC coefficients. We use the first 26 MFCC coefficients for classification, and we justify this choice in Experiments section. The intuition behind choosing the first few coefficients is that most natural signals typically have exponential drop in energy for increasingly higher frequency [14] [15]. The coefficients are then averaged over all windows (in this case 100 windows) to produce the feature vectors of size 26 for every phoneme. We normalize the individual coefficients across all the phonemes using z-normalization.

3.3 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 22,050 samples per second. 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 12,400, 20,750 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 12,400, 10,375 and 10,200 which are very close to one another.

3.4 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 [16] 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 Fig. 3 where the word `talk` is misaligned. The phoneme `/T/` is completely missed as silence, and the vowel `/ɔː/` is segmented into `/T/` and `/ɔ/`. The true number of such erroneous cases is not possible to determine on our dataset because it needs manual examination of all 376,509 phonemes; however, the authors of [1] 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 including most of the words in English. However, there exist some proper nouns and compound words that are absent in the dictionary such as `Quakerism` and `Donatist`. This limitation is not limiting our classification algorithm as we can append the phonetic spellings of these words over time.

4 DTW-based phoneme classification

We use K-nearest neighbor algorithm (K-NN) for classification using dynamic time warping (DTW) distance. There exist a number of studies that find DTW as the most competitive distance measure for time-series data [17]. For completeness, we provide the definition of DTW here. Let us define two signals $x = x_1, x_2, \ldots, x_n$ and $y = y_1, y_2, \ldots, y_m$ of length n and m where $n > m$ without losing generality.

$$\begin{aligned}
DTW(x, y) &= D(n, m)\\
D(i, j) &= (x_i - y_j)^2 + min \begin{cases} D(i-1, j)\\ D(i, j-1)\\ D(i-1, j-1)\end{cases}\\
D(0, 0) &= 0\\
\forall_{ij} \quad D(i, 0) &= D(0, j) = \infty
\end{aligned}$$

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 [18]). 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 $w \geq |n - m|$ 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 [18], early abandoning technique [19], hardware acceleration technique [20], summarization technique [21], anticipatory pruning [22], 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 5 (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 as follows:

$$\begin{aligned}
\forall i \quad U_i &= max(x(i - w : i + w))\\
\forall i \quad L_i &= min(x(i - w : i + w))\\
LB\_Keogh &= \sum_{i=1}^{n} \begin{cases} (y_i - U_i)^2 & \text{if } y_i > U_i\\ (y_i - L_i)^2 & \text{if } y_i < L_i\\ 0 & \text{otherwise}\end{cases}
\end{aligned}$$

LB_Keogh and many other recent optimization techniques for DTW-based similarity search are only defined for equi-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 6 (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

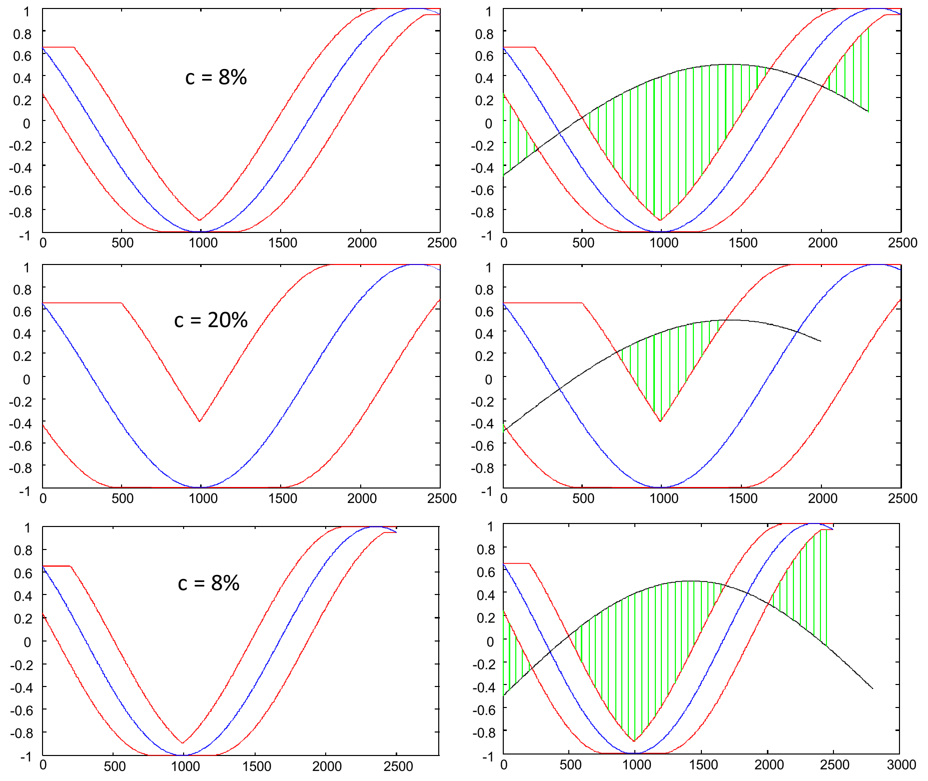


Fig. 5 (*left*) Envelopes U and L for the candidate (*blue*) sinusoid over different windows. (*Right*) *LB_Keogh* using the prefix of the longer signal. (Top) The query shown in black is the shorter of the signals of length 2300. Note that the difference in length between the blue and the black signals 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 (color figure online)

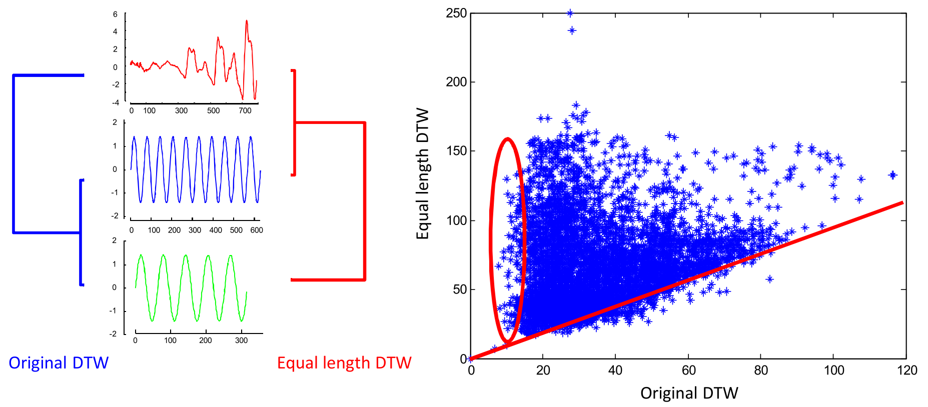


Fig. 6 (*Left*) Example of erroneous association caused by DTW distances on resampled equi-length signals. (*Right*) The scatter plot shows there exist numerous other cases of similar signals being measured as dissimilar by equi-length DTW

Table 1 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

distances with and without resampling and create a scatter plot. Figure 6 (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.

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, Fig. 5 (right) shows the prefixed *LB_Keogh* or in short, *PLB_Keogh* computation that ignores all the $x_i, i = m + 1, \ldots, n$. 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* [18] and is omitted here.

In addition to *PLB_Keogh*, we can use the early abandoning DTW computation that 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 1 shows a list of techniques from [19] and their applicability in phoneme classification.

As we have a set of techniques applicable to our problem, 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 do not 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 $c = 0.3$ in our experiments and justify this choice in 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 700–1000 than to signals of length 1000–1300 for $c = 30\,\%$. It will not be fair if we pick the nearest neighbor without a measure for this. Some previous works [23] 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

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

```
Ensure: Return the labels for the signals of the Test
 1: Preprocessing
 2: for $x \leftarrow Train_i$ , $i = 1, 2, \ldots, |Train|$ do
 3:     Calculate $U_x$ and $L_x$ using $w_x \leftarrow \lfloor c * |x| \rfloor$
 4: end for
 5: Testing
 6: $d_K \leftarrow \infty$
 7: for $y \leftarrow Test_j$ , $j = 1, 2, \ldots, |Test|$ do
 8:     Calculate $U_y$ and $L_y$ using $w_y \leftarrow \lfloor c * |y| \rfloor$
 9:     for $x \leftarrow Train_i$ , $i = 1, 2, \ldots, |Train|$ do
10:         $LB \leftarrow \infty$
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 $d < d_K$ and $j \geq k$ then
21:                 $d_K \leftarrow d$, Save $x$ as one of the $k$ neighbors
22:             end if
23:         end if
24:     end for
25:     Use the K-NNs to find the phoneme class
26: end for
```

the classification process in the Algorithm 1. The value of K is set to 7 for all our experiments and is justified in Experiments section.

5 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 whether 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 Fig. 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 of the phoneme hierarchy 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 [6] using MFCC features. Using multiple classification techniques at different hierarchy has also been done [8,24]. 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

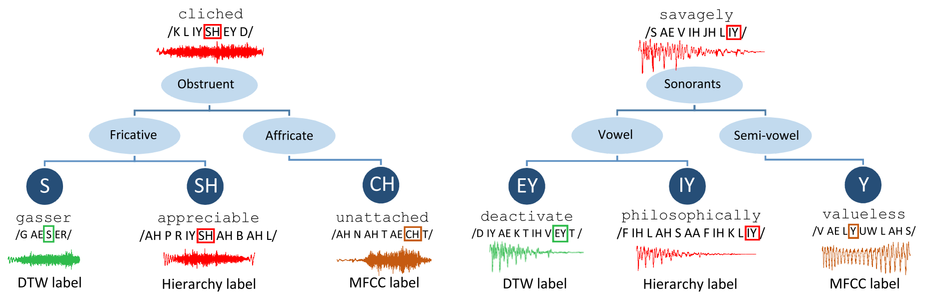


Fig. 7 Two examples. Each tree shows the unknown phoneme, its waveform and the source word at the root. The three leaf nodes of a tree show the three classes the algorithms produce. In each of the leaf nodes, the nearest neighbor phoneme, its waveform and the source word are shown

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 Experiments section.

To describe the idea more clearly, we show two examples in Fig. 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.

6 Improved vowel classification

In Sect. 5, we describe our dual-domain classification techniques that achieve reasonably good performance on a 39-class problem. We analyze the confusion matrix on the test data to understand the critical classes for our classifier. We find that vowels have the highest miss-classification rate in our classification approach. Fifty-seven percent of all misclassifications are vowel phonemes misclassified as other phonemes including vowels and non-vowels. Since the complete confusion matrix has many rows and columns, we prefer to illustrate it by a heat map. The vertical axis in the heat map is the true class, and the horizontal axis is the predicted class. Each cell (r, c) has the number of phonemes in class r classified as class c divided by the number of all phonemes in class r. All the yellow and orange segments in Figs. 8 and 9 except the diagonal show classification errors.[1] Since green (light) means zero and red (dark) means one in the heat map, we expect to see green color everywhere off the diagonal and red color on the diagonal for an ideal classifier. The vowel square at the bottom left has the most off-diagonal values. We identify that the vowels are produced without any

[1] Color figures are available in online version of the paper.

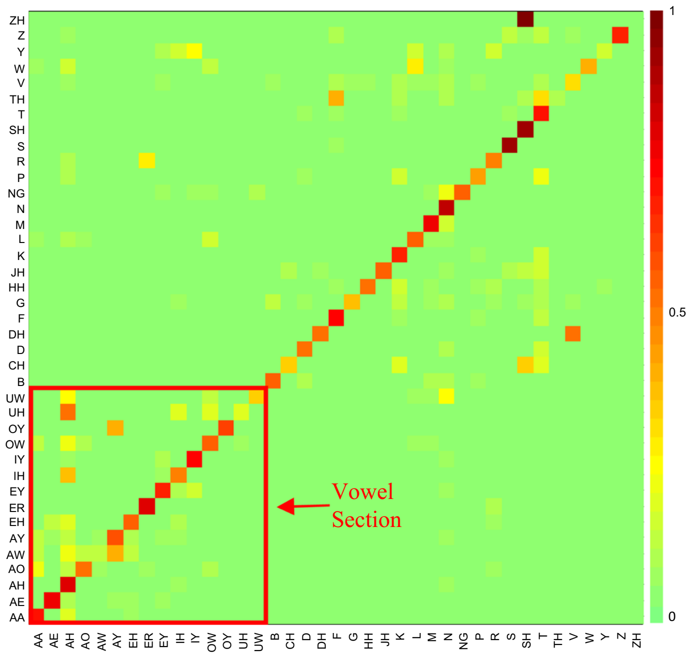


Fig. 8 Normalized confusion matrix of phoneme classification

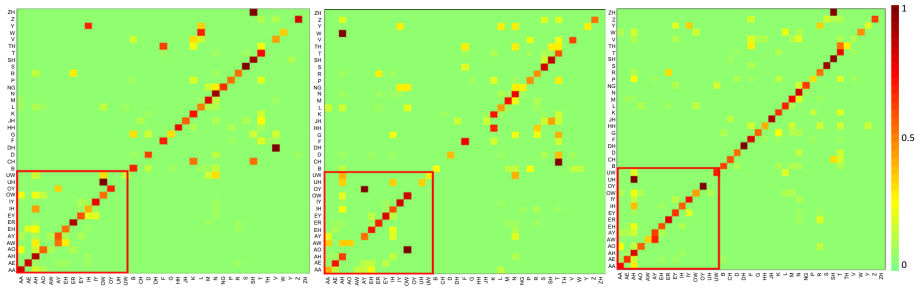


Fig. 9 Normalized confusion matrices of phoneme classification for each test set separately. (*Left*) **GG** (*middle*) **OO** and (*right*) **WW**

obstacle in the vocal tract and therefore they merge with consonants very well in natural speech. Forced alignment error for vowels is large compared with consonants which results in more confusion.

Typically, MFCC coefficients are computed over sliding windows of 25 ms with 10 ms overlap [13]. Thus, phonemes are converted to a sequence of MFCC coefficients, and we take average over the sequence for each coefficient to come up with a fixed length feature vector. Our observation is that vowels need more temporal information for classification, and we change the classifier at the vowel node to use DTW distances on the temporal sequences of MFCC coefficients. Finally, at the time of comparison, we take average of the distances from each coefficient. This approach just moves the averaging over time to averaging over distances, which preserves temporal information more.

7 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 [25], and it is publicly available. Unlike others, we have also uploaded the massive distance matrix of size $3000 \times 376,509$ for interested researchers.

As mentioned earlier, we have dictionary words from three sources. We have 376,509 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. We have three testing sets (**GG**, **OO** and **WW**), each of which has 1000 samples. We hold all these 3000 testing samples out of the training set. We use all the other phonemes (373,509 phonemes) in our dataset as training set. We then use the same training set to classify samples in the testing set from all three dictionaries. For a test set of 1000 phonemes, it takes 1 week to calculate the DTW distances and the classification accuracy. We have limited resource which is not sufficient to perform a k-fold cross-validation. A test set of 3000 phonemes is the largest set we could classify in reasonable time which resulted in a disproportional train-test split. Cross-validation on such large training data using such expensive algorithm is not possible. The number of neighbors and the size of the window are 7 and 30 %, respectively.

Competitor Algorithms There exists a variety 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 [6]. 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 [25]. We use the abbreviation of the title of the paper, OAHPC, as the name of the algorithm in the subsequent sections.

The PhoneRec [26] 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 I ER S IH M T IH G K T/` where the correct phoneme transcription is `/EH G OH S EH N T R IH K/`. Another example, the word `greasy` is converted to the phoneme sequence `/HH UW IY Z IY T/` where the correct phoneme transcription is `/G R IY S IY/`. We use a set of 200 phonemes, and the accuracy is less than 10 %. We do not 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 [9]. We were unable to reproduce from the paper, and there was no code and documentation available from the authors. Recall TIMIT [3] is a manually labeled data and it costs at least $250 to obtain license for the data.

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 Fig. 2). We use DTW-based

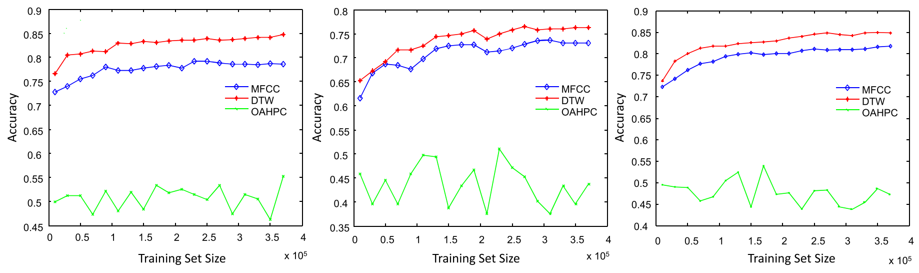


Fig. 10 Incremental accuracy of the algorithms in the *top* two layers of the hierarchy on three test sets. (*Left*) **GG** (*middle*) **OO** and (*right*) **WW**

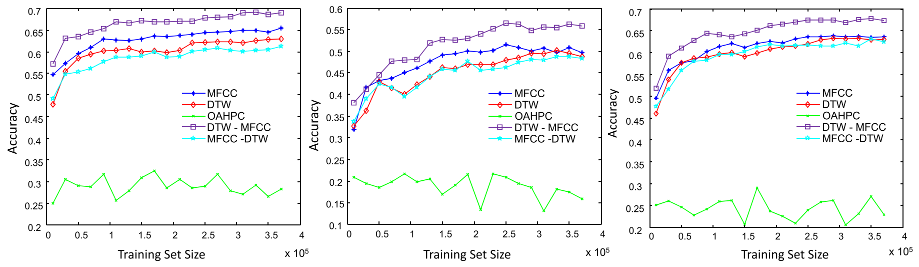


Fig. 11 Incremental accuracy of the algorithms in whole of the hierarchy on three test sets. (*Left*) **GG** (*middle*) **OO** and (*right*) **WW**

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 Fig. 10. 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.

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 376,509 phonemes. The results are shown in Fig. 11. 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.

Adding the vowel classifier The approach explained in Sect. 6 preserves more temporal information. Thus, by using more temporal information in the distance calculation, we can achieve more accurate classification in comparison with average MFCC-based K-NN. We gain 64 % accuracy rate in vowel classification using our base method. We improve it to

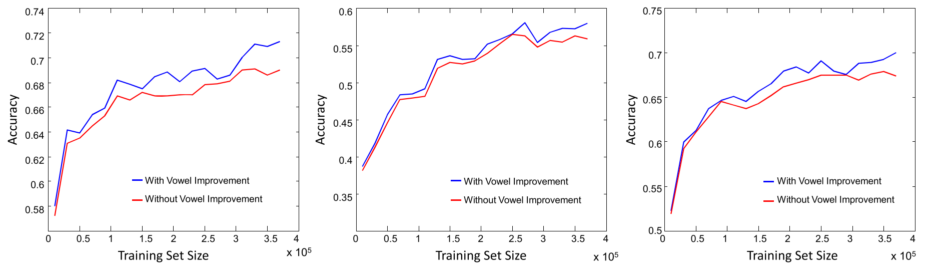


Fig. 12 Accuracy of our classifier with and without improved vowel classification. (*Left*) **GG** (*middle*) **OO** and (*right*) **WW**

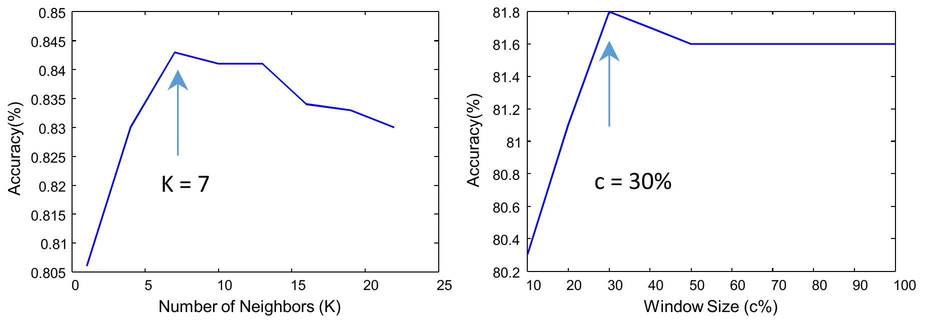


Fig. 13 (*Left*) accuracies for different K at the top two layers. (*Right*) accuracies for different window sizes (c) at the *top* two layers

74 % using the method mentioned in Sect. 6. This 10 % gain in the vowel node is equal to a roughly 3 % accuracy gain overall because more than one-third of the phonemes in the dataset are vowels. Comparison between the accuracy of our classifier with and without this improvement is shown in Fig. 12.

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. Testing all possible parameter combinations is expensive. Therefore, we choose a training set of 60,000 phonemes randomly from the original training set of 373,509 phonemes. We use the same test set of 3000 phonemes and measure the accuracy for different parameter combination. The results are shown in Fig. 13.

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

Scalability We adopt all the techniques from [19] 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 two times speedup over the original implementation without any optimization. Figure 14 (left) shows the speedup curve. We achieve the best speedup from early abandoning technique. The reason why we cannot achieve the dramatic speedup as [19] is explainable. Since we cannot resample, the

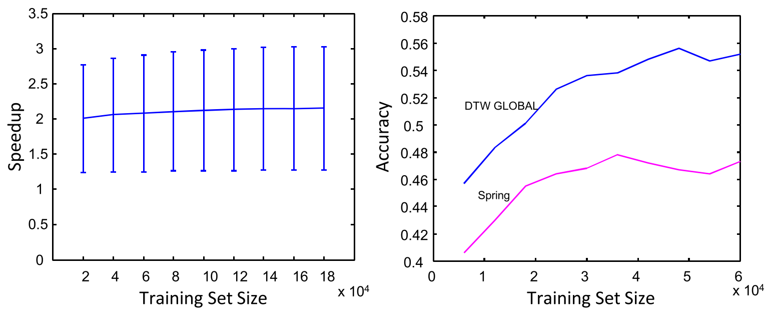


Fig. 14 (*Left*) speedups achieved using the techniques in Table 1 (*right*) comparison between Spring-based [28] classifier and global DTW-based K-NN classifier. The former requires no segmentation and the later uses Forced Aligner

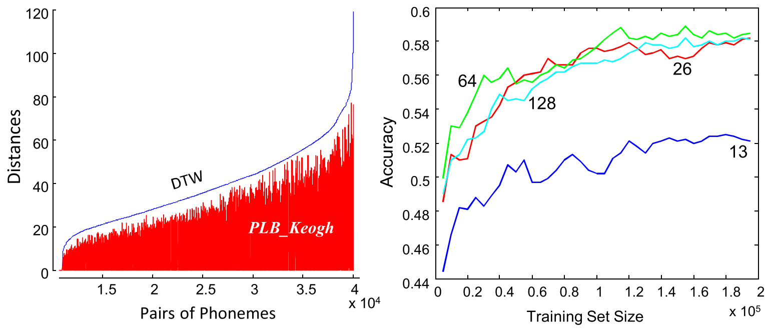


Fig. 15 (*Left*) visual proof for *PLB_Keogh* bound. (*Right*) sensitivity of the number of MFCC coefficients used for the classifiers

window size has to be larger than the length difference. As shown in Fig. 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* [27] which is a very weak bound. Figure 15 shows the visual proof of *PLB_Keogh* working for 40,000 pairs of signals of different lengths where there is no counter example found.

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 [28] 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 in Fig. 14 (right). The biggest drawback of the above algorithm is that a phoneme can match in between two phonemes in a word. This added

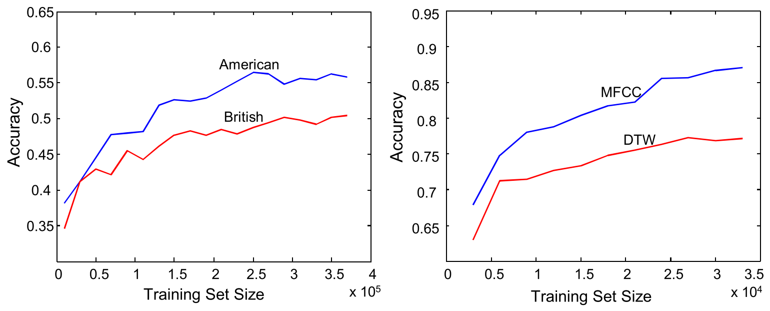


Fig. 16 (*Left*) Accented phoneme classification accuracies using the whole dataset. (*Right*) accent detection using K-NN classifiers

confusion hurts the accuracy significantly, and thus, it supports the use of Forced Aligner for segmentation before doing the phoneme classification.

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 Fig. 15 (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.

8 Case studies

8.1 Accented phoneme classification

Accent is a type of variation in the spoken form of a language. In this case study, we investigate whether 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 16 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 are 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.

We further investigate whether 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 two-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.

8.2 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

Table 2 Speaker-invariant accuracy using dual-domain hierarchy

Testset	GG	OO	WW	Except Testset
GG	–	38.5	45.7	**46.8**
OO	30.5	–	37.5	**39.3**
WW	43	38.5	–	**47.0**

Bold numbers are the maximum of each row in the table

Table 3 Speaker-invariant accuracy using OAHPC

Testset	GG	OO	WW	Except Testset
GG	–	14.2	19.0	**19.9**
OO	**20.8**	–	13	15.8
WW	**24.9**	15.8	–	18.6

Bold numbers are the maximum of each row in the table

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.

Tables 2 and 3 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 againts 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.

8.3 Linguistic analysis of errors

In addition to improve our classifier at vowel node, we analyze the confusions made by our method from linguistic perspective. There are some confusions that are justifiable in the sense that humans often confuse these phonemes. This essentially translates to a weighted error computation which treats some confusions more severe than others [6]. Some examples are as follows:

(1) `/EH/, /IH/, /UH/` and `/AW/` are classified as `/AH/` in several cases. The reason is that these phonemes are so similar to `/AH/` that it is really hard even for humans to classify them correctly by listening to a sub-second audio sample.
(2) `/ER/` is identified with 81 % accuracy; the majority of errors are labeled as `/R/`. These confusions are probably not errors but reflect transcription errors or differences in transcription style in the dictionary.
(3) `/IH/` is identified with 49 % accuracy. It is confused most often with spectrally similar lax vowel `/AH/`, and this confusion is often made also by human listeners.
(4) `/IY/` is identified with 74 % accuracy (again same is true for human listeners). It is most often confused with the spectrally similar vowel `/EY/` which has an `/IY/`-like off-glide at the end.
(5) `/AA/` is identified with 69 % accuracy. Confusions with `/AO/` should probably not be regarded as errors because these vowels are merged into one vowel in many dialects of American English.

(6) /EY/ is identified with 66 % accuracy. It is confused with /IY/. The off-glide portion of /EY/ is spectrally similar to /IY/.

Different level of discounting can improve the overall accuracy at various rates. Note that discounting errors always improves accuracy. A simple discounting is to weight the misclassifications for phoneme pairs inside the sets of the above list 50 % of that for general misclassifications. We experiment using 50 % discounting rate and achieve 77 % accuracy. Since we have not experimented with other discounting rates, we do not include linguistic discounting techniques in overall classification accuracy. We show this case study to demonstrate that linguistic properties can change the interpretation of the classification accuracy.

9 Recognizing word from sequence of phonemes

Until now we talk about phoneme classification approaches. Our major motivation is that a phoneme classifier can play an important role in robust speech recognition. We view a robust speech recognizer in a discrete format as a pipeline of independent pieces: segmenter, classifier and recognizer. A segmenter finds the phoneme boundaries, a classifier converts the phonemes to discrete symbols, and a recognizer arranges the phoneme sequence in a meaningful word. The great benefit of a pipelined recognition procedure is that each stage can use independent prior knowledge to correct errors from the previous stage. As we see in Fig. 3, there exists large segmentation error because of forced alignment which our classifier is resilient with. In this section, we show a simple experiment to show that the errors from our classifier can also be corrected in the recognizer stage.

We can correct some phoneme classification errors in recognition step because not all the sequences of phonemes can be mapped to a meaningful English word. For example, the word `cold` is made up of `/K OW L D/`, and if we classify all the phonemes in this word, we may end up with a sequence `/K IY L D/`. Since the output is not a valid sequence of phonemes in our dictionary, it must be corrected. So, we can use one or multiple steps to replace some phonemes by simply finding the most similar word to this sequence in our dictionary. We may find more than one word which are similar to the detected sequence (in our example both `cold: /K OW L D/` and `yield: /Y IY L D/` are similar to `/K IY L D/`, and we need to use some other information to choose one. One of the most important information is the context of the sentence. Another source of information is the frequency of the word in written or spoken English. More complex prior can be generated from linguistics and speech knowledge.

We make the simplest choice to go for the most frequent word and ignore larger context in the sentence level. We find all candidate phoneme sequences for a given phoneme sequence by using edit distances between them. Edit distance is a way of quantifying how dissimilar two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other [29]. Although edit distance is computed in $O(nm)$ where n and m are the number of elements in the two given sequences, the number of phonemes in each word is very small, so this calculation can be done very quickly despite the quadratic run time. For example, the edit distance between `/K OW L D/` and `/K IY L D/` is one since one can transform the former to the latter by replacing `/OW/` by `/IY/`. After we find all the candidates by using edit distance, the one with the highest frequency in English is picked as the final output. We use word frequencies reported in [30]. The whole process is shown in Fig. 17.

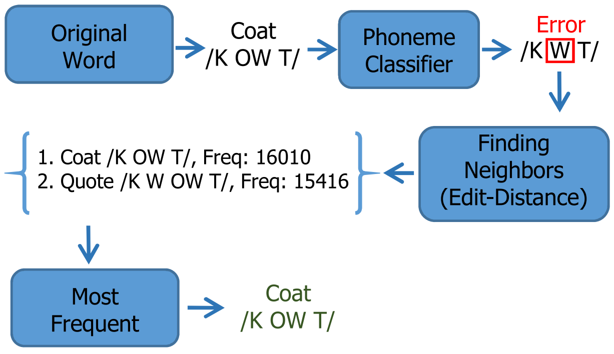

Original Word
Coat
/K OW T/
Phoneme Classifier
Error
/K W T/
Finding Neighbors (Edit-Distance)
1. Coat /K OW T/, Freq: 16010
2. Quote /K W OW T/, Freq: 15416
Most Frequent
Coat
/K OW T/

Fig. 17 Detecting word from a given sequence of phonemes

Fig. 18 Comparison between phoneme-level accuracy and word-level accuracy. Some phoneme-level errors can be corrected while converting sequence of phoneme to a word

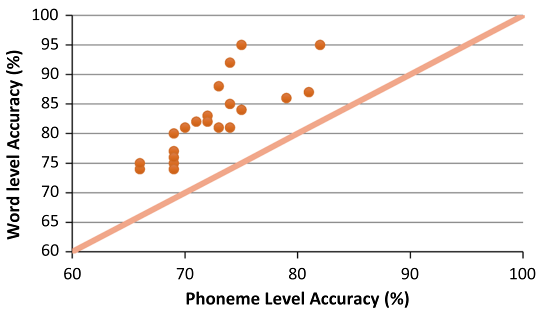


Table 4 Comparison between Google speech recognizer and our proposed method of generating words out of sequence of phonemes

GMU Speaker ID	Original sentence	Phoneme classification accuracy (%)	Our result	Google recognizer
Arabic 18	We also need a small plastic snake and a big toy frog for the kids	60	We also need a small plastic scow and a big toy frog for the kids	We also need a small plastic snake toy for golden keys
Arabic 37	And maybe a snack for her brother Bob	60	And maybe a week for were brother Bob	And maybe it is not for Boulevard
Arabic 26	She can scoop these things into three red bags	56	She can scoop these things into said red bags	She can scoop those things and the three you right back

We do different experiments to see whether this method can give us more accurate result compared with phoneme classification accuracy. We picked 24 sets of English words with 30 words uniformly at random from our dataset, run all the steps explained in Fig. 17 on them and compare the phoneme classification accuracy with the word detection accuracy on each set. Results are shown in Fig. 18. We correct up to 80 % of phoneme classification errors in the recognition process, and since all the points are above the diagonal, we are always able to correct some errors of phoneme classification in phoneme sequence recognition.

We do another experiment to compare our method of generating words with a voice recognition application in real world. We choose Google speech recognizer which is available both online and as a built-in application on Android devices. It converts the individual parts of speech to words, and then (or at the same time), it uses a language model to produce the most meaningful sentence out of those words. For this experiment, we have taken three audio samples where subjects say three different sentences collected from the GMU speech accent archive [31]. All three speakers have Arabic accent. We segment all the sentences into phonemes and use our classifier to classify them. We then use our word recognizer to convert the sequence of recognized phonemes to words. To compare our results with a real application output, we use Google recognizer to convert exactly same sentences into text. You can see the results in Table 4. Although in this experiment our phoneme classifier cannot give us as accurate results as it does for native English speakers, the word recognition system still does an acceptable job. Our results are closer to the original sentence than the Google recognizer's results in all three cases. Google recognizer is trained for native speakers so it cannot recognize accented speech well, but our recognizer can generate closer sentences because it can correct some phoneme classification errors in word recognition step.

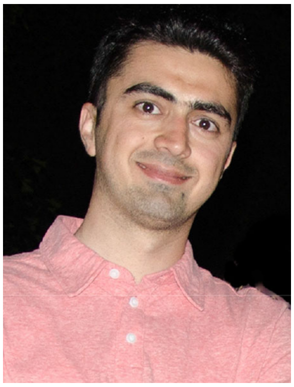
10 Conclusion

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 a problem. We use a novel dataset of 376,509 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. Yuan J, Liberman M (2008) Speaker identification on the scotus corpus. In: Proceedings of acoustics 2008
2. Hamooni H, Mueen A (2014) Dual-domain hierarchical classification of phonetic time series. In: ICDM
3. Garofolo J (1993) Timit acoustic-phonetic continuous speech corpusldc93s1, web download. Philadelphia: linguistic data consortium
4. International phonetic alphabet. https://en.wikipedia.org/wiki/International_Phonetic_Alphabet
5. Lee K-F, Hon H-W (1989) Speaker-independent phone recognition using hidden Markov models, acoustics, speech and signal processing. IEEE Transa on 37(11):1641–1648
6. Dekel O, Keshet J, Singer Y (2005) An online algorithm for hierarchical phoneme classification. In: Proceedings of the first international conference on machine learning for multimodal interaction, ser. MLMI'04, 2005, pp 146–158
7. Carla L, Fernando P (2011) Phoneme recognition on the timit database. Speech Technologies, [Online]. http://www.intechopen.com/books/export/citation/BibTex/speech-technologies/phoneme-recognition-on-the-timit-database
8. Schwarz P, Matejka P, Cernocky J (2006) Hierarchical structures of neural networks for phoneme recognition. In: 2006 IEEE international conference on acoustics, speech and signal processing, 2006. ICASSP 2006 proceedings
9. Rahman-Mohamed A, Dahl GE, Hinton GE (2012) Acoustic modeling using deep belief networks. IEEE Trans Audio Speech Lang Process 20(1):14–22

10. Salomon J (2001) Support vector machines for phoneme classification, Master of Science, School of Artificial Intelligence, Division of Informatics, University of Edinburgh
11. Mohamed A, Hinton G (2010) Phone recognition using restricted Boltzmann machines. In: 2010 IEEE international conference on acoustics speech and signal processing (ICASSP), pp 4354–4357
12. Dewey E (1970) Godfrey, relative frequency of english spellings. In: International Reading Association, Anaheim, California, May 6–9, 1970. http://files.eric.ed.gov/fulltext/ED042572.pdf
13. Matlab implementation to compute mel frequency cepstrum coefficients. http://www.mathworks.com/matlabcentral/fileexchange/32849-htk-mfcc-matlab/content/mfcc/mfcc.m
14. Mueen A, Nath S, Liu J (2010) Fast approximate correlation for massive time-series data. In: SIGMOD conference, pp 171–182
15. Faloutsos C, Ranganathan M, Manolopoulos Y (1994) Fast subsequence matching in time-series databases. SIGMOD Rec 23:419–429
16. The cmu pronouncing dictionary. http://www.speech.cs.cmu.edu/cgi-bin/cmudict
17. Ding H, Trajcevski G, Wang X, Keogh E (2008) Querying and mining of time series data: Experimental comparison of representations and distance measures. In: Proceedings of the 34 th VLDB, pp 1542–1552
18. Keogh E (2002) Exact indexing of dynamic time warping. In: Proceedings of the 28th international conference on very large data bases, ser. VLDB '02, pp 406–417
19. Rakthanmanon T, Campana B, Mueen A, Batista G, Westover B, Zhu Q, Zakaria J, Keogh E (2012) Searching and mining trillions of time series subsequences under dynamic time warping, ser. KDD '12, pp 262–270
20. Sart D, Mueen A, Najjar W, Niennattrakul V, Keogh EJ (2010) Accelerating dynamic time warping subsequence search with gpus and fpgas. In: ICDM, pp 1001–1006
21. Petitjean F, Ketterlin A, Ganarski P (2011) A global averaging method for dynamic time warping, with applications to clustering. Pattern Recognit 44(3):678–693
22. Assent I, Wichterich M, Krieger R, Kremer H, Seidl T (2009) Anticipatory DTW for efficient similarity search in time series databases. Proc VLDB Endow 2(1):826–837
23. Mueen A (2013) Enumeration of time series motifs of all lengths. In: ICDM, pp 547–556
24. Cesa-Bianchi N, Gentile C, Zaniboni L (2006) Hierarchical classification: combining Bayes with svm. In: Proceedings of the 23rd international conference on machine learning, ser. ICML '06, pp 177–184
25. Repository for supporting materials. http://cs.unm.edu/~hamooni/papers/Dual_2014/index.html
26. Phoneme recognizer based on long temporal context, http://speech.fit.vutbr.cz/software/phoneme-recognizer-based-long-temporal-context
27. Yi B-K, Jagadish HV, Faloutsos C (1998) Efficient retrieval of similar time sequences under time warping. In: Proceedings of the fourteenth international conference on data engineering, Orlando, Florida, USA, 23-27 Feb 1998, pp 201–208
28. Sakurai Y, Faloutsos C, Yamamuro M (2007) Stream monitoring under the time warping distance. In: 2013 IEEE 29th international conference on data engineering (ICDE), vol. 0, pp 1046–1055
29. Edit distance tutorial. https://web.stanford.edu/class/cs124/lec/med.pdf
30. Word frequency data. corpus of contemporary american english. http://www.wordfrequency.info
31. Gmu speech accent archive. http://www.accent.gmu.edu



Hossein Hamooni is a Ph.D. student of computer science at the University of New Mexico. He earned M.S. and B.S. degrees from Amirkabir University of Technology (Tehran Polytechnich) in Tehran, Iran. His research interest includes time series mining with applications to human speech and social media.

Abdullah Mueen is an Assistant Professor of computer science in the University of New Mexico. He obtained Ph.D. degree in 2012 from the University of California, Riverside. Prior to that, he obtained B.Sc. degree from Bangladesh University of Engineering Technology. Dr. Mueen's research interest in large scale heterogeneous data mining with a focus on spatio-temporal data. He has won runner-up in the dissertation contest at SIGKDD 2012 and won the best paper award in the same conference. He has published articles in top data mining venues including KDD, ICDM, SDM and SIGMOD.

Amy Neel received her master's degree in speech-language pathology from the University of Oklahoma, spent ten years as a clinical speech-language pathologist, and received her Ph.D. in speech and hearing science and cognitive science from Indiana University in 1998. Her research focuses on intelligibility in normal speech and in dysarthric speakers, including those with Parkinson disease and oculopharyngeal muscular dystrophy.