



# **Prosodic features in state-of-the-art spoken language identification**

*Sam Sucik*

**MInf Project (Part 1) Report**

Master of Informatics  
School of Informatics  
University of Edinburgh

2019



# Abstract

TO-DO

## **Acknowledgements**

Thanks to Paul Moore for productive collaboration while building the baseline system, to Steve Renals for his supervision and optimism, and to David Snyder for his advice.

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Motivation . . . . .	7
1.2	Aims . . . . .	7
1.3	Contributions . . . . .	7
<b>2</b>	<b>Background</b>	<b>9</b>
2.1	The task: spoken language recognition . . . . .	9
2.2	Shallow utterance-level approach to LID: i-vectors . . . . .	10
2.3	Less shallow approach: d-vectors . . . . .	11
2.4	Deep utterance-level approach: x-vectors . . . . .	12
2.5	Features used in language identification . . . . .	14
2.5.1	Acoustic features . . . . .	15
2.5.2	Prosodic features . . . . .	16
2.5.3	The continuous Kaldi pitch as a prosodic feature . . . . .	18
2.5.4	Illustrating the prosodic features used in this work . . . . .	19
<b>3</b>	<b>Data</b>	<b>21</b>
3.1	The corpus: GlobalPhone . . . . .	21
3.2	Data partitioning . . . . .	22
3.3	Data preprocessing . . . . .	24
3.4	Dealing with invalid data . . . . .	25
3.5	Overview of the data used . . . . .	25
<b>4</b>	<b>Implementation</b>	<b>29</b>
4.1	The Kaldi toolkit . . . . .	29
4.2	Kaldi recipes used in this work . . . . .	30
4.3	Choice of classifier . . . . .	31
4.4	Final architecture . . . . .	32
4.5	Computing environment . . . . .	34
4.6	Hyperparameters . . . . .	34
<b>5</b>	<b>Experiments</b>	<b>35</b>
5.1	Baseline . . . . .	35
5.2	Shifted delta cepstra . . . . .	35
5.3	KaldiPitch+Energy vectors . . . . .	35
5.4	MFCC/SDC + KaldiPitch+Energy . . . . .	36

5.5	Fusion of MFCC/SDC and KaldiPitch+Energy scores . . . . .	36
5.6	Timeline for the experiments . . . . .	36
<b>6</b>	<b>Results</b>	<b>37</b>
<b>7</b>	<b>Discussion</b>	<b>39</b>
<b>8</b>	<b>Future work</b>	<b>41</b>
8.1	Plans for Part 2 of the MInf project . . . . .	41
8.2	Other future ideas . . . . .	41
<b>9</b>	<b>Conclusions</b>	<b>43</b>
	<b>Bibliography</b>	<b>45</b>

# Chapter 1

## Introduction

### 1.1 Motivation

LID is useful in ASR, in voice assistants, emergency call routing, etc. Traditionally, acoustic features are used (influence of ASR on LID and SID). Prosodic LID is much rarer, although results show that prosodic information can help identify language (Lin and Wang, 2005), and that both LID and ASR can benefit from using acoustic *and* prosodic features (González et al., 2013; Ghahremani et al., 2014). Just last year, a novel architecture for LID utilising *x-vectors* was proposed by Snyder et al. (2018a), dramatically improving the state-of-the-art results. Although the authors find that using bottleneck features from an ASR DNN yields better results than using the standard acoustic MFCC features, even the ASR DNN was trained just using MFCCs. Thus the work ignores the potential of speech information other than that captured by MFCCs.

### 1.2 Aims

In this work, I aim to reproduce the state-of-the-art x-vector LID system and explore the use of prosodic features in addition to acoustic ones. Because the system uses a relatively novel architecture, in which a TDNN aggregates information across a speech segment, I also compare two types of acoustic features, one which has such aggregation over time encoded (SDC) and one that only contains information about single frames (vanilla MFCC).

### 1.3 Contributions

1. Adapted an existing x-vector speaker verification implementation (based on Snyder et al. (2018b)) for language identification

2. Explored the choice of classifiers and chose a different one than Snyder et al. (2018a)
3. Prepared the Global Phone corpus for LID with the x-vector system, extending the original partitioning of the corpus into datasets and analysing invalid data
4. Built and evaluated a baseline, end-to-end x-vector LID system using 19 languages of the Global Phone corpus
5. Explored, set and tuned important hyperparameters of the system, mainly the number of training epochs of the x-vector TDNN
6. Researched literature concerning the use of acoustic and prosodic features in language identification, speaker verification and ASR
7. Designed, run and evaluated experiments comparing two types of acoustic features (MFCC and SDC) and two prosodic features (pitch, energy), and their combinations
8. Built a system which has the potential to be open-sourced as part of the Kaldi ASR toolkit, to be used by a wider community



# Chapter 2

## Background

This chapter elaborates on the key concepts relevant to my work, as shown in this condensed description of the project: Exploring **spoken language identification** in the context of the recently proposed **x-vector** system (contrasted with the more established state-of-the-art **i-vector** approach, followed by the more novel **d-vector** systems), focusing on utilising **prosodic information** in addition to the standard **acoustic information**.

### 2.1 The task: spoken language recognition

Spoken language recognition means recognising the language of a speech segment. The task is similar to speaker recognition and, in the past, similar systems have been used for the two tasks. Importantly, recognition is typically realised as one of two different tasks:

- Identification (multi-class classification): answering the question "For a speech segment  $X$ , which one (from a number of supported targets) is its target (language or speaker)  $T_x$ ?"
- Verification (binary classification): "Is  $T_x$  the target (language or speaker) of the speech segment  $X$ ?"

Identification is more suitable for use cases with a small and stable set of possible targets – such as the set of supported languages. There, computing the probability of  $T_x$  being each of the target languages is feasible. Verification, on the other hand, is more suitable for cases where the set of possible targets less constrained – such as the large and changing set of possible speakers in speaker verification systems. There, it is often infeasible to compute the probability distribution over all possible values of  $T_x$ ; instead, the system typically focuses only on evaluating the probability of  $T_x$  being the hypothesised speaker. Throughout this work, I focus on *language identification* (LID) with a *closed set* of target languages (i.e. not including the option to identify a speech segment's language as unknown/other).

## 2.2 Shallow utterance-level approach to LID: i-vectors

This approach, with its numerous variations, has now been the state of the art for 8 years – since first introduced by Dehak et al. (2011) for speaker recognition and later applied by Martinez et al. (2011) in language recognition.

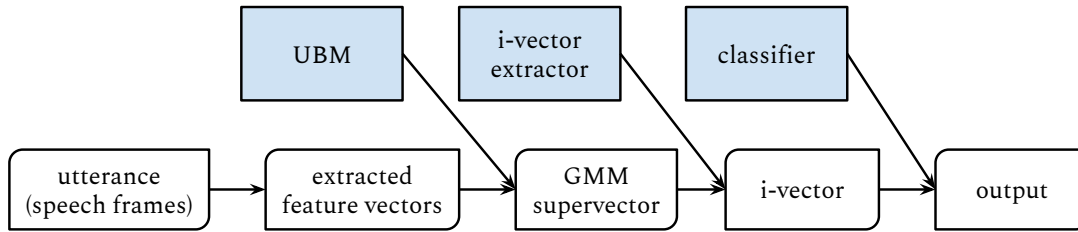


Figure 2.1: Language identification using a typical i-vector system.

The main components of a typical i-vector system, together with their use for prediction, are shown in Fig. 2.1. The universal background model (UBM) is a Gaussian mixture model (GMM) consisting of a large number (e.g. 2048) multivariate mixture components. The UBM is trained using the Expectation-Maximisation algorithm to model the observation space of frame-level feature vectors  $\mathbf{x}$  computed from all training utterances (typically 39-dimensional vector using the MFCC features, see Section 2.5.1). Given the trained UBM, an utterance  $X$  can be represented by a language- and channel-specific GMM supervector  $M$  as:

$$M_X = m + Tw_X \quad (2.1)$$

Here,  $m$  is the language- and channel-independent GMM supervector of the UBM (consisting of the stacked means of all mixture components).  $T$  is the *total variability matrix* – also called the i-vector extractor, and  $w_x$  is the *identity vector* (i-vector) of sample  $X$ . The total variability matrix  $T$  is “a matrix of bases spanning the subspace covering the important [language and channel] variability in the supervector space” (Martinez et al., p.862). Effectively,  $T$  projects between the high-dimensional GMM supervector space and the low-dimensional *total variability subspace* (also called the *i-vector space*). The i-vector  $w_x$  is then a low-dimensional set of factors projecting the supervector onto the total variability subspace base vectors. The i-vector extractor  $T$  is trained again using Expectation-Maximisation (for details see Mak and Chien (2016, p. 100)). Without providing too much detail, I highlight the aspect that is most relevant to my work: training  $T$  is based on calculating and further combining the 0<sup>th</sup>, 1<sup>st</sup> and 2<sup>nd</sup> order statistics which are computed by *summing over the frames* of an utterance.

Once the i-vector extractor has been trained, any utterance can be represented by its i-vector. This enables training a relatively simple and fast classifier operating over the low-dimensional i-vectors. Different classifiers have been successfully used with the i-vector back end; for example, Martinez et al. initially tried using these, all achieving roughly equal performance:

1. a linear generative model – modelling the i-vectors of each language by a Gaussian, with a shared full covariance matrix across all language-modelling Gaussians
2. a linear Support Vector Machine computing scores for all languages by doing binary classification in the one-versus-all fashion
3. a two-class logistic regression also doing one-versus-all classification.

At test time, a sequence of feature vectors for utterance  $X$  is projected using the UBM into the high-dimensional *supervector space*, producing  $M_X$ . Then,  $T$  is used to extract the utterance-level i-vector, which is processed by the classifier.

Despite producing utterance-level scores, I describe the i-vector pipeline as shallow because it aggregates frame-level information over time very early (when projecting  $X$  into the supervector space), effectively treating an utterance as a bag of frames and disregarding any temporal patterns spanning over multiple frames.

## 2.3 Less shallow approach: d-vectors

Introduced for speaker verification by Variani et al. (2014) and later adapted and applied to language identification by Tkachenko et al. (2016), this approach uses neural networks to extract frame-level information, which is then aggregated across frames to produce utterance- or language-level vectors. Importantly, nothing changes about the final classification stage itself: It is the differences in producing the vector representations what makes i-vectors, d-vectors and x-vectors differ from each other.

The biggest change introduced in d-vector systems compared to i-vectors is the notion of frame-level processing while considering each frame's temporal context, i.e. the sequence of a few preceding and following frames. While Variani et al. achieve this by simply feeding the feature vectors of the neighbouring frames together with the frame of interest into a standard deep neural network, Tkachenko et al. use a more suited architecture commonly used for processing temporal sequences in automatic speech recognition: the *time-delay neural network* (TDNN).

A TDNN works like a convolutional neural network (CNN) with 1-dimensional convolutions: only along the time axis, as opposed to the more common 2-dimensional convolutions in image CNNs. Fig. 2.2 shows a TDNN with three layers which processes information over 9-frame contexts. Note that each blue circle from any layer corresponds to the layer itself (a collection of neurons, i.e. *convolutional kernels*), and all blue circles drawn above each other as corresponding to a particular layer are in fact just one circle – the layer itself – sliding over multiple inputs. For example, the convolutional kernels from layer 1 are first applied to feature vectors  $\mathbf{x}_1$ - $\mathbf{x}_5$ , then to vectors  $\mathbf{x}_3$ - $\mathbf{x}_7$  and then to  $\mathbf{x}_5$ - $\mathbf{x}_9$ . Notice how the network is made more sparse by using *subsampling*, i.e. not using the connections drawn in light grey. A concise and commonly used description of the sketched architecture is shown in Tab. 2.1 (notice the difference between the interval and set notation); "layer context" meant relative to the preceding layer.

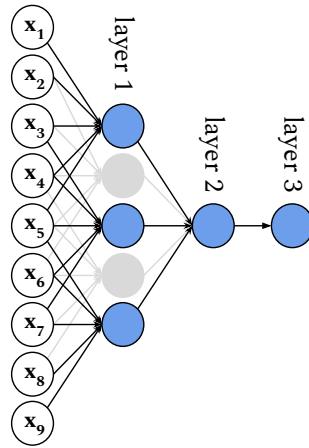


Figure 2.2: Example of a time-delay neural network.

	LAYER CONTEXT	TOTAL CONTEXT
LAYER 1	$[t - 2, t + 2]$	5
LAYER 2	$\{t - 2, t, t + 2\}$	9
LAYER 3	$\{t\}$	9

Table 2.1: Example of architecture description of a TDNN (corresponds to Fig. 2.2).

In a d-vector system, a TDNN is trained to do direct frame-wise language identification; for this purpose, an additional softmax layer can be added to produce classification outputs. Often, between the convolutional layers and the output layer the architecture contains a few fully connected layers. After training, the classification layer is disregarded and a frame is instead represented by the activation values from the last hidden layer. The TDNN thus serves as a feature extractor, and the representations produced are referred to as *embeddings*. A d-vector representing an entire utterance is then simply the average of the embeddings of all frames from the given utterance.

Despite the naive averaging, utterances are no longer treated merely as bags of frames because each embeddings contains features extracted over short temporal window: 21-frame windows in the case of Tkachenko et al. and 41-frame windows used by Variani et al., making d-vectors less shallow than i-vectors.

## 2.4 Deep utterance-level approach: x-vectors

The x-vector approach, introduced last year by the John Hopkins University team first for speaker verification (Snyder et al., 2018b) and subsequently for language identification (Snyder et al., 2018a), can be viewed as an extension to d-vectors, producing utterance-level embeddings even for variable-length speech segments. The architecture consists of (see also Fig. 2.3):

1. the context-aggregating TDNN layers operating at frame level (with the final context window of  $\pm 7$  frames),

2. a *statistics pooling layer* which computes the mean and the standard deviation over all frames, effectively changing a variable-length sequence of frame-level activations into a fix-length vector, and
3. an utterance-level part consisting of 2 fully connected bottleneck layers which extract more sophisticated features and compress the information into a lower-dimensional space, and an additional softmax output layer.

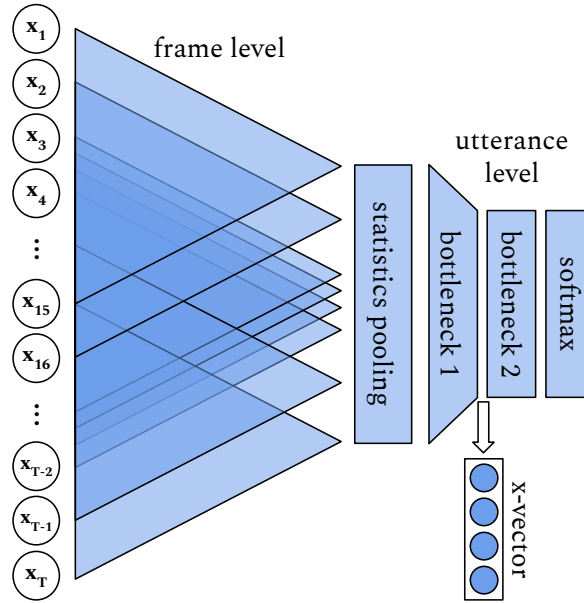


Figure 2.3: A high-level sketch of the x-vector TDNN from Snyder et al..

After the TDNN has been trained, embeddings extracted as the activations of the first bottleneck layer – named x-vectors – are then utterance-level representations and can be used directly as input for a final classifier. The biggest difference compared to the d-vector neural networks discussed in Section 2.3 is that the x-vector TDNN is trained to do not frame-level, but utterance-level classification. The utterance-level statistics (mean and standard deviation) are computed *as part of the network* and are further processed by the additional fully connected layers with non-linear activation functions. These bottleneck layers also make it possible to preserve a higher number of useful features after the statistics-pooling layer: By enabling the last frame-level layer to inflate the representations to 1500-D. The bottleneck layers then learn a transform back to the 512-dimensional space (high-dimensional vectors would be inconvenient for the final classifier) while extracting from the aggregated statistics the features most useful for utterance-level classification (not just for frame-level decisions).

Because of being an utterance-level classifier, the x-vector TDNN can be used directly as an end-to-end system, although Snyder et al. found that extracting x-vectors and training a separate classifier to classify them yielded slightly better results. Perhaps an even stronger argument in favour of the two-stage process is that, by using an external light-weight classifier, one can easily change the set of supported languages without having to expensively re-train the TDNN. Snyder et al. showed that simply re-training the classifier is enough to gain very good results for languages the TDNN has not

		LAYER CONTEXT	TOTAL CONTEXT	NUMBER OF UNITS
FRAME LEVEL	LAYER 1	$[t - 2, t + 2]$	5	512
	LAYER 2	$\{t - 2, t, t + 2\}$	9	512
	LAYER 3	$\{t - 3, t, t + 3\}$	15	512
	LAYER 4	$\{t\}$	15	512
	LAYER 5	$\{t\}$	15	1500
STATISTICS POOLING LAYER		$[1, T]$	UTTERANCE	3000-DIMENSIONAL OUTPUT
UTTER. LEVEL	BOTTLENECK 1	N/A	UTTERANCE	512
	BOTTLENECK 2	N/A	UTTERANCE	512
	SOFTMAX	N/A	UTTERANCE	L-DIMENSIONAL OUTPUT

Table 2.2: Description of the x-vector TDNN from Snyder et al. and used in this work.  $T$  is the number of frames in a given utterance. Table adapted from Snyder et al. (2018a, p. 106).

observed during training (although the results are indeed worse than those for observed languages).

The x-vector system consistently beat several state-of-the-art i-vector architectures, which is a particularly interesting finding because i-vector systems have for long been dominant despite the nowadays so "fashionable" and successful deep learning approaches. Even so, the 2-stage x-vector system using a simple classifier still dominates the end-to-end neural network alternative.

## 2.5 Features used in language identification

Historically, automatic speech recognition (ASR) has been the most important area of speech processing and it has driven forward other areas including language and speaker recognition. In particular, LID systems still exist mostly as part of ASR systems where the language of speech needs to be identified before attempting to transcribe the speech. Hence, data preprocessing and input feature types for LID systems often come from the ASR area. In particular, the feature extraction processes follow the ASR objectives: To extract the language- and speaker-*independent* information important for discriminating between different phonemes produced by a speaker's vocal tract. The feature types devised for this aim are termed *acoustic features*. However, these features can also be characterised by the information they disregard or at least to not capture explicitly: the language- and/or speaker-*dependent* information: intonation, stress, pitch range, tone, and others, collectively referred to as *prosodic information*.

As this work focuses on exploiting prosodic information to improve LID, it is desirable to understand both acoustic and prosodic features: how they differ, why is prosody

useful and how can prosodic information be modelled in systems that process frame sequences rather than isolated speech frames.

### 2.5.1 Acoustic features

The most commonly used acoustic feature type are the Mel-frequency cepstral coefficients (MFCCs). These are coefficients that aim to characterise the shape of the vocal tract, which corresponds to characterising the spectral envelope of produced sounds or, in a way, to the actual phones produced. Without providing too much technical detail, I remind the reader of the main steps in computing MFCCs in Fig. 2.4 (for details, see for example Chapter 10 of Holmes and Holmes (2002)).

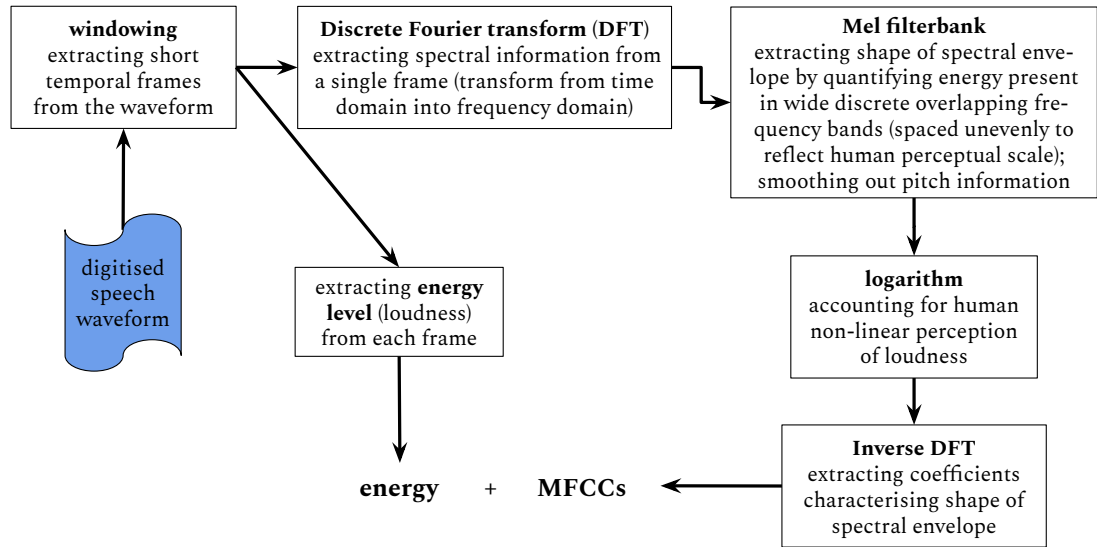


Figure 2.4: Main steps in computing the MFCC features including energy. Adapted from Shimodaira and Renals (2019, p. 10).

To capture local temporal trends going beyond the single frame level, computed features such as MFCCs are often augmented with frame-specific *dynamic* features – *delta cepstra* (also just *deltas* or  $\Delta$ ) and double deltas ( $\Delta\Delta$ ). These are simple approximations of the first and second derivatives of the original acoustic feature, and are computed for frame  $t$  like this ( $\mathbf{x}_t$  is the MFCC feature vector):

$$\Delta(t) = \mathbf{x}_{t+1} - \mathbf{x}_{t-1}, \quad \Delta\Delta(t) = \Delta(t+1) - \Delta(t-1) \quad (2.2)$$

using the context of  $\pm 2$  frames. Such dynamic features are justified especially in systems which apply the bag-of-frames approach, not extracting temporal trends (such as the standard i-vector architectures).

For modelling even longer-range trends specifically in the LID task, Torres-Carrasquillo et al. (2002) used an extension of simple deltas called the *shifted delta cepstra* (SDC) features – essentially, deltas computed at multiple points and concatenated with the

original MFCC vector as previously. SDCs can be configured by setting their 4 parameters:  $N$ ,  $d$ ,  $p$ ,  $k$  corresponding to a  $N$ - $d$ - $p$ - $k$ , a typical one being 7-1-3-7.  $N$  refers to the number of cepstral coefficients taken into account (not necessarily the full MFCC vector),  $p$  denotes the distance between consecutive points at which deltas are computed,  $d$  determines the size of the window for computing deltas ( $d = 1$  corresponds to Eq. 2.2) and  $k$  is the number of points at which to compute deltas. The SDCs added to the original MFCC vector are then of the form:

$$\Delta_{SDC}(t + ip) = \mathbf{x}_{t+ip+d} - \mathbf{x}_{t+ip-d} \quad \forall i \in \{0, \dots, k-1\} \quad (2.3)$$

As with standard deltas, capturing long-range trends with SDCs can certainly be beneficial for i-vector systems. Ferrer et al. (2016) used an i-vector system with SDC features as the state-of-the-art baseline system for their novel experiments. Additionally – following the successful application of SDCs by Torres-Carrasquillo et al. in a pre-i-vector system, Sarma et al. (2018) report slightly better results with SDCs compared to using high-resolution MFCCs when training an i-vector architecture with a deep neural network UBM. Still, the usefulness of SDCs is not so clear in architectures inherently able to extract temporal trends.

Compared to the prosodic features described next, the discussed classic acoustic features have one clear advantage where the LID system is part of a bigger ASR pipelines: The two systems can share the same feature vectors without the need to compute additional features solely for the purpose of language identification.

## 2.5.2 Prosodic features

Prosody is about information other than that directly describing the produced phones; the standard elements of prosody are: intonation/tone, stress and rhythm (Prieto and Roseano, 2018). As discussed below, importantly, each of these elements can be used to differentiate between languages.

- *Intonation* and *tone* are perceived qualities typically associated with the variation of the physical quality called pitch, i.e. the fundamental frequency of vibration of the vocal folds ( $F_0$ ). The term 'tone' is used where pitch variation is contrastive (i.e. changes to it discriminate between different words): in the so-called *tonal languages*, such as Chinese, Vietnamese and Thai. In each of these languages, a small 'alphabet' of distinct tone levels or tone contours can be observed as it conveys meaning. Intonation, on the other hand, denotes the non-contrastive pitch variation mostly associated with the speaker's emotions, language, dialect, social background, speaking style, and other factors (see, for example, the study by Grabe (2004) on the intonational differences between different British English dialects).
- *Stress* (also termed *accent*), despite being a somewhat ambiguous term, generally refers to the relative emphasis on certain syllables within a word or words within a sentence, realised as a variation in loudness (*dynamic accent*), pitch (*pitch accent*) or vowel length (*quantitative accent*). Specific *lexical* (intra-word, i.e.



syllable-level) stress patterns or rules are often characteristic of a language. For instance, the within-word location of stress is fixed but different for Czech and Polish (*fixed stress*, see Goedemans and van der Hulst (2013a)), or describable by a set of rules in Arabic (*rule-based stress*, see Goedemans and van der Hulst (2013b)).

- *Rhythm* refers to regularity in sub-word unit lengths, with the most recognised theory of *isochrony* categorising languages into three types based on their rhythm: *syllable-timed* where all syllables have roughly equal durations (e.g. French or Turkish), *mora-timed* where all moras<sup>1</sup> have equal durations (e.g. Slovak or Japanese), and *stress-timed* languages, e.g. German and Arabic (here, the intervals between stressed syllables are of equal durations).

Before turning to bespoke alternative features for modelling prosody, it should be noted that some prosodic aspects can be captured by acoustic features like MFCCs. Because prosody surfaces as temporal variation, systems which extract information from frame sequences (such as TDNN-based systems) have the potential to capture it. MFCCs can be (and often are) used together with energy (see Fig. 2.4), which enables capturing dynamic accent. Even without using energy as part of MFCCs, the quantitative accent should be possible to capture because durations of vowels are preserved as a result of windowing extracting the frames at evenly spaced points of a speech waveform. Regarding rhythm, the duration regularity (isochrony) should be possible to capture for syllable-timed and mora-timed languages because syllable boundaries occur at phoneme boundaries, which are captured by acoustic features. For stress-timed languages, however, the units with equal durations are marked by stressed syllables, meaning that this regularity may not be fully captured – in particular if an energy measure is not included in MFCCs, or if the stress is realised as a variation in pitch rather than in loudness. Considering the prosodic aspects that acoustic features alone *could* theoretically capture, I conclude that the most important prosodic feature that can be deemed complementary to acoustic features and added to the input is pitch: to enable modelling pitch accent, tone, and language-specific intonation.

Perhaps unsurprisingly, extracting information from pitch for LID has been explored in the past – although the number of studies is very small compared to those using the conventional acoustic features. Long before i-vectors, Lin and Wang (2005) attempted to do language identification solely from the pitch contour. This was segmented into syllable-like units (around 100-200ms), each contour segment subsequently represented as the first few coefficients of its approximation by Legendre polynomials. Even though the results were far from those achieved by today’s systems, it was demonstrated that LID can be done on pitch information alone. Later, Metze et al. (2013) showed improvements in ASR on tonal (and partly on non-tonal) languages when augmenting MFCCs with custom-designed  $F_0$  variation features. Nevertheless, one disadvantage of using pitch persisted: The fact that  $F_0$  is typically considered to be only present in voiced sounds and undefined for unvoiced ones – making a pitch contour based on  $F_0$  discontinuous. This inconvenience has been recently addressed by Ghahre-

---

<sup>1</sup>Mora is a basic timing unit; a long syllable having two moras and a short syllable having one mora. See Crystal (2008, p. 312) for a broader definition and discussion.

mani et al. (2014) who developed a pitch-tracking algorithm that approximates pitch values even for unvoiced regions and produces a continuous pitch contour. Adding this feature to MFCCs, the authors report improved ASR performance mainly on tonal languages. To the best of my knowledge, this feature type has not yet been used for language identification.

In this work, I explore two feature types that explicitly help to capture prosodic information: *energy* and *pitch*. I refer to these as the prosodic features.

- By energy I mean the logarithm<sup>2</sup> of the raw energy extracted in the MFCC pipeline right after windowing (as shown in Fig. 2.4). To isolate the effects of using this prosodic feature, I do not include it in any of the acoustic features used in this work. However, I do not exclude the 0<sup>th</sup> cepstral coefficient from MFCCs, even though it bears some loudness-related information; more specifically, it can be considered as "a collection of average energies of each frequency bands in the signal that is being analyzed" (Zheng and Zhang, 2000).
- By pitch I mean the continuous pitch (and features derived from it) extracted using the Kaldi pitch algorithm presented by Ghahremani et al.. Because this feature type is not well known, I elaborate on it in the next section.

### 2.5.3 The continuous Kaldi pitch as a prosodic feature

Even though the original paper (Ghahremani et al., 2014) contains a detailed description of the Kaldi pitch algorithm (along with a working implementation as part of the Kaldi ASR toolkit), I provide an additional, more holistic view to give the reader an intuitive understanding of the algorithm and the features it produces (see Fig. 2.5). Perhaps the biggest contribution lies in abandoning the binary decision making about voiced/unvoiced speech, and only calculating pitch for the voiced frames. Instead, soft decisions are made based on the values of the normalised cross-correlation function (NCCF). For two pieces of digitised waveform separated by temporal distance  $l$  from each other and represented as vectors  $\mathbf{v}_{t,0}$  and  $\mathbf{v}_{t,l}$ , the NCCF is computed as follows (adapted from Ghahremani et al., Eq. 2):

$$\Phi_{t,l} = \frac{\mathbf{v}_{t,0}^T \mathbf{v}_{t,l}}{\sqrt{(\mathbf{v}_{t,0}^T \mathbf{v}_{t,0})(\mathbf{v}_{t,l}^T \mathbf{v}_{t,l}) + B}} \quad (2.4)$$

where normally  $B = 0$ , but the authors use a non-zero value to push the NCCF values towards 0 where the cross-correlation on its own is very low. By computing NCCF for different spacings  $l$  – called *lags* by the authors, and being effectively the periods of different hypothesised frequencies – one obtains continuous confidence values about the presence of the corresponding frequencies in the signal for the time position  $t$ . After computing these for all temporal positions, the algorithm finds the optimal

---

<sup>2</sup>The use of logarithm is given solely by the Kaldi toolkit I use for feature extraction, although it is by no means an arbitrary design decision: Human perception of loudness is non-linear and logarithm is used also in the standard MFCC computation pipeline (see Fig. 2.4).

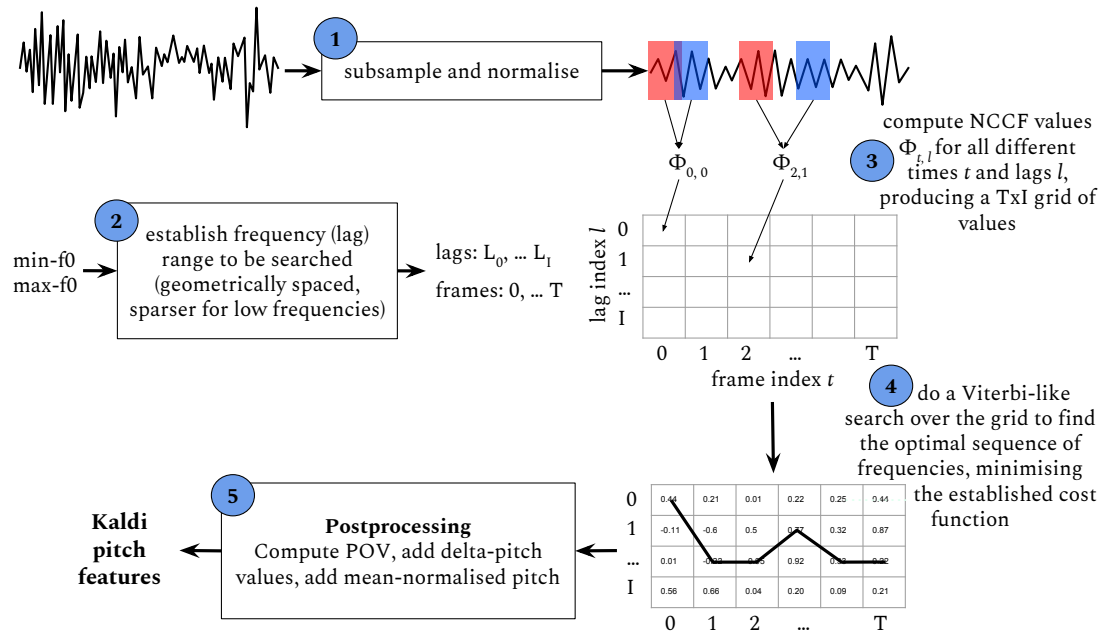


Figure 2.5: The steps of the Kaldi pitch algorithm.

continuous contour by minimising the proposed cost function (see Ghahremani et al., Eq. 4), which prefers higher NCCF values and penalises too low frequencies and too big frequency jumps.

Finally, the algorithm post-processes the raw pitch contour to provide these additional features:

1. A mean-normalised pitch contour, with the mean computed over 151-frame windows and weighted by a measure roughly corresponding to the log likelihood of a particular frame being voiced (i.e. putting more weight on voiced frames).
2. *Probability of voicing* (POV): A feature which is not a probability, but acts similarly and has been empirically found by the authors to provide improvements in ASR performance.
3. Delta-pitch terms, computed in the same way as the simple deltas in Eq. 2.2.

In this work, I work with all four feature types (the raw pitch contour and the three derived feature types) and refer to them collectively as the Kaldi pitch.

### 2.5.4 Illustrating the prosodic features used in this work

Having familiarised the reader with the two prosodic features I will use (the energy and the Kaldi pitch), I now illustrate them on a realistic example. Because the multi-lingual corpus used for this project does not contain English, I used the freely accessible sample utterance from the CSR-I (WSJ0) Complete corpus<sup>3</sup>, which has important

<sup>3</sup><https://catalog.ldc.upenn.edu/LDC93S6A>

characteristics such as the recording conditions and the speaking style the same as the corpus I later use (see Section 3.1).

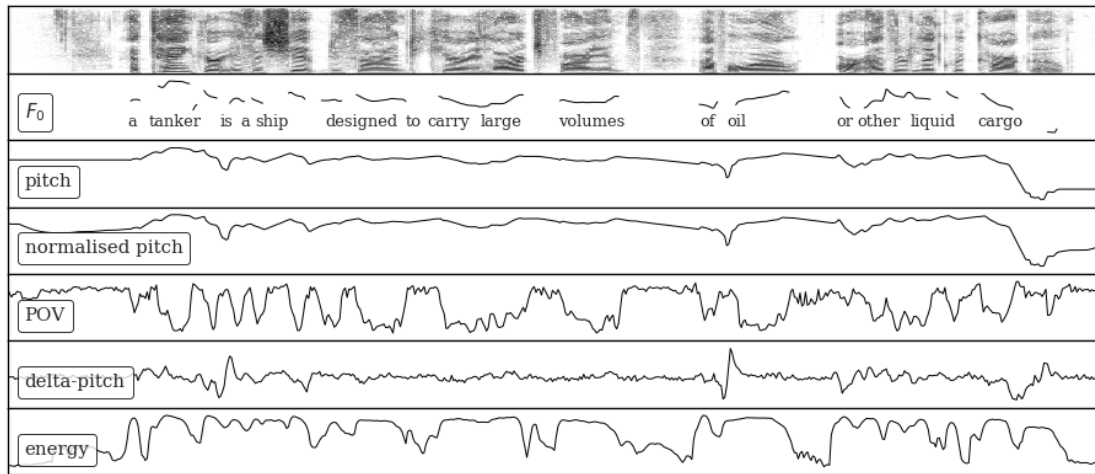


Figure 2.6: The prosodic features I use in this project (together with the spectrogram and the fundamental frequency  $F_0$ ), illustrated on an English utterance. I manually created the time-aligned transcription for better understanding.

Fig. 2.6 shows the prosodic features on the English utterance. For clarity and comparison, I added the spectrogram as well as the discontinuous  $F_0$  contour – both generated in Praat (Boersma and Weenink, 2019). As expected, the  $F_0$  segments match the Kaldi pitch contour. Notice how the Kaldi pitch does not drop in unvoiced regions, but rather behaves like a smooth bridging between the regions. Also note that the POV values are effectively negated: High values corresponding to the regions where true  $F_0$  is absent and low values marking the voiced regions.

# Chapter 3

## Data

Because LID systems are often part of ASR systems, it makes sense to use the same datasets to train both. Additionally, however, the U.S. National Institute of Standards and Technology (NIST) has been organising dedicated Language Recognition Evaluation<sup>1</sup> (LRE) challenges since 1996, providing multilingual datasets which are today the standard benchmark for evaluating language recognition systems (used also by Snyder et al. (2018a)). The NIST datasets, however, only focus on narrowband (8kHz) conversational telephone speech and include strong channel variability as a result of the uncontrolled recording environment, which makes it more difficult to analyse any observed effects and attribute them to particular language pair similarities or particular prosodic aspects. In this work, I use a relatively compact ( $\sim 400$  hours of speech) corpus which is in many aspects different from the NIST datasets.

### 3.1 The corpus: GlobalPhone

The corpus I use is the GlobalPhone multilingual ASR corpus (Schultz et al., 2013), more precisely its newest version – updated in 2016 and covering 22 languages from around the world (see Fig. 3.1): Arabic (AR), Bulgarian (BG), Chinese-Mandarin (CH), Chinese-Shanghai (WU), Croatian (CR), Czech (CZ), French (FR), German (GE), Hausa (HA), Japanese (JA), Korean (KO), Brazilian Portuguese (PO), Polish, (PL), Russian, (RU), Spanish, (SP), Swahili (SWA), Swedish (SW), Tamil (TA), Thai (TH), Turkish (TU), Ukrainian (UA), and Vietnamese (VN). The corpus follows the style established for English by the Wall Street Journal-based corpora (Paul and Baker, 1992) such as the CSR-I (WSJ0) Complete corpus<sup>2</sup>. It consists of newspaper articles read by native speakers (around 100 speakers per language, each speaker producing about 20 minutes of audio by reading 3-5 articles), recorded as wideband (16kHz) audio in a controlled environment, using a close-talking microphone (in the case of GlobalPhone the Sennheiser microphone HD-440-6).

---

<sup>1</sup><https://www.nist.gov/itl/iad/mig/language-recognition>

<sup>2</sup><https://catalog.ldc.upenn.edu/LDC93S6A>



Figure 3.1: Languages of GlobalPhone as shown roughly at the location where they were recorded. Greyed out are Hausa, Swahili and Ukrainian – the three languages I did not use in this work due to corrupt data (see Section 3.4). Also notice that Russian was, in fact, recorded in Belarus, and Spanish in Costa Rica.

The controlled environment, speaking style and nature of the texts (all are major newspaper articles) make the data better suited for analysing effects of, for example, prosodic features: Any observed trends can be more reliably attributed to my controlled independent variables (such as the feature type) or to the known language differences. Another advantage of the GlobalPhone corpus is the rich vocabulary (compare with short telephone conversations in the NIST corpora). Finally, from a linguistic view, the corpus covers a wide range of language families and prosody-based categories, while also providing an interesting sample of related languages in the form of 5 Slavic languages (which the author – as a native speaker of Slovak – can relate to): Bulgarian, Czech, Croatian, Russian, and Ukrainian.

Unfortunately, the corpus does not cover spontaneous speech (which – as known in ASR – is much more challenging than read text). The extremely low channel variability is also a tight constraint and it is possible that, even if prosodic features improve LID performance on this dataset, the results will not necessarily generalise well to more realistic scenarios with greater channel variability and noise.

## 3.2 Data partitioning

GlobalPhone ships with a partitioning of each language’s data into training, evaluation and testing (in GlobalPhone documentation referred to as training, development and evaluation, respectively) sets, with the sizes of the 3 partitions being roughly 80%, 10%, 10%, respectively. However, for the purposes of the x-vector architecture, a 4-way partitioning is required:

1. training set – for training the x-vector TDNN,
2. enrollment set – for training the x-vector classifier,
3. evaluation set – for tuning the hyperparameters of both the TDNN and the classifier,
4. testing set – for final end-to-end evaluation on unseen data.

To create the desired 4-way partitioning, I allocated part of the training data for enrollment. This way, I preserved the original evaluation and testing, enabling fairer comparisons of my results with any other future works that use the GlobalPhone corpus. Because the classifier is typically a much simpler model (with many fewer learnable parameters) than the x-vector TDNN, I allocate only 1/8 of the training set for enrollment (splitting each language’s data in equal proportion). This way, my new training set accounts for roughly 70% of the whole corpus, while the 3 other sets are roughly 10% each.

Importantly, GlobalPhone<sup>3</sup> still misses partitioning for certain languages:

1. no partitioning for Czech, Polish, Tamil, Swahili, Ukrainian, Vietnamese and Wu,
2. incomplete partitioning for Arabic (incomplete evaluation and test sets), and for French, Japanese, Russian (missing evaluation sets for all three languages).

Note that a partitioning in this context is represented simply in the form of partition- and language-specific lists of speakers (identified by their speaker IDs). The ASR GlobalPhone Kaldi recipe<sup>4</sup> extends the original GlobalPhone partitioning to also cover Czech, French, Japanese, Polish, Russian, Swahili and Vietnamese. I utilise this partitioning in order to stay consistent with previous work using the recipe. For the rest of the languages with incomplete or missing partitioning (Arabic, Tamil, Ukrainian and Wu), I partitioned the data myself, following the same approach as the GlobalPhone authors – the only imposed constraint being: ”no speaker appears in more than one group [partition] and no article is read by two speakers from different groups” (Schultz, 2002, p. 348). This constraint, however, could only be satisfied for languages which contained speaker metadata information (i.e. which articles were read by the particular speaker). Because this metadata was missing for Bulgarian, French, German, Polish, Tamil, Thai and Vietnamese, the splitting of the original training set into training and enrollment portions (and, in the case of Tamil, the entire 4-way partitioning) for these languages was done solely on a random basis.

I automated the entire partitioning process such that it would never allow one speaker to appear in multiple sets. Further, the way I understand the declared 80%-10%-10% splitting is that it is the *numbers of speakers* for each languages that are split following this ratio (since there is roughly constant amount of data per speaker). Therefore, the way I realise the desired 4-way partitioning is based on distributing speakers in the desired 70%-10%-10%-10% ratio. Unfortunately, I had to somewhat relax the constraint

<sup>3</sup>as of newest version of its documentation I had access to: v4.1 from January 2016

<sup>4</sup><https://github.com/kaldi-asr/kaldi/tree/master/egs/gp/s5>

that the same article cannot appear in multiple sets. My implementation attempts to construct such ideal partitioning, but – if it cannot be constructed – the allowed number of article overlaps is iteratively increased (from 0, incrementing by 1) until a satisfying partitioning is found. From all the languages for which the partitioning was generated non-randomly, the following languages had non-zero article overlaps: Arabic (385), Turkish (16), and Wu (49).

For the final partitioning (after excluding languages and speakers with corrupt data, see the next section) is shown in Tab. 3.2.

### 3.3 Data preprocessing

These are the preprocessing steps that were carried out either only once (the time-consuming ones), or repeatedly (once for each of my experiments) but without any changes. Although the Kaldi GlobalPhone recipe was used as a starting point, in the end, most of the used Shell scripts were heavily adapted or re-written from scratch.

**Converting data to .wav format** was the most time-consuming preprocessing step (taking more than 7 hours) and was done only once. The reason for this step is that the Kaldi toolkit requires .wav files for computing features, but the GlobalPhone audio files are in the Shorten (.shn) format. The conversion itself was done using the `shorten` and `sox` command-line tools. Even though this step could have been made a part of the feature computation pipeline itself – without the need to store all the .wav files – it would be time-consuming as the conversion would have to be repeated.

**Organising the data into partitions** was done symbolically (without re-organising the .wav files), hence taking very short. This made it possible for the step to be a part of the pipeline repeated separately for each experiment – or each time an utterance or a speaker was found to be unusable due to corrupt data and had to be excluded from the partitioning. Because the previous step did not create any data partitioning (i.e. all .wav files were only grouped by languages), in this step, for each partition, a list referring to the .wav files for that partition’s speakers across all languages was compiled (based on the files storing the partitioning described in Section 3.2). In addition to this list (named conventionally `vaw.scp`), a number of accompanying lists was generated as required by the conventions of Kaldi: `utt2spk` (mapping utterances to speakers), `utt2lang` (mapping utterances to languages), `utt2len` (mapping utterances to their lengths), and a number of further lists derived from these essential ones.

**Splitting longer utterances into shorter segments** was done to match the set-up used by Snyder et al. (2018a): Training set utterances were not split in this step because the x-vector TDNN splits them into chunks of at most 1000 frames (10s) at training time (TO-DO: reference the appropriate implementation section for details). Enrollment set utterances were split uniformly into segments of length 30s, and evaluation and testing set utterances were split into 10s segments (and later also into 3s segments because the trained system in each experiment was additionally tested on 3s segments). Note that I did not discard the cut-offs from the utterance ends (even though they were shorter than the desired lengths). Another downside of the uniform cutting is that



utterances were split unnaturally – not necessarily on breaks, hence creating arbitrary edge boundaries not naturally found in speech. The reason why I chose to implement this way of splitting the utterances was its simplicity; in the future, this would be one of the design decisions I would change.

**Computing voice activity decision (VAD)** for all segments to identify and later discard the frames containing silence. This was computed using the VAD implementation found in Kaldi, using the parameter values from the speaker recognition x-vector recipe<sup>5</sup>, in particular the log-energy threshold of 5.5. Because in my experiments I wanted to frame-wise concatenate feature vectors of different feature types, I needed to make sure I would always discard the very same set of frames for all feature types. Hence, I computed VAD only once – on the MFCC feature vectors – and re-used it to discard silence frames throughout all experiments.

### 3.4 Dealing with invalid data

While carrying out the preprocessing steps, a lot of data was identified as corrupt:

- while converting .shn to .wav, converting all utterances for Hausa, Swahili and Ukrainian failed with the "No magic number" error thrown by the `shorten` tool<sup>6</sup>
- the following utterances were not converted due to the same error: 095\_29 for Bulgarian; 003\_2 for German; almost all utterances for speakers 015, 136, and up to 3 utterances for speakers 014, 017, 021, 022, 024, 026, 031, 036, 058, 139 in Portuguese; utterance 087\_102 in Russian; utterance 007\_77 in Turkish; and utterance 084\_122 for Vietnamese
- while computing VAD, the `compute-vad` binary throwing the "Empty feature matrix for utterance [utterance-ID]" error for utterances 058\_16 and 058\_18 in Portuguese

Naturally, the utterances which could not be converted were discarded. I also discarded the Portuguese speakers 015 and 136 altogether. Most importantly, the three problematic languages – Hausa, Swahili and Ukrainian – were not used any further, only the 19 remaining languages.

### 3.5 Overview of the data used

I present an overview of the data I ultimately used in my experiments – after discarding the invalid data. Tab. 3.2 shows the speakers contained in each partition so that my

<sup>5</sup><https://github.com/kaldi-asr/kaldi/tree/master/egs/sre16/v2>, this is the recipe I adapted for LID in my implementation of the x-vector system.

<sup>6</sup>reported also here: <https://wiki.inf.ed.ac.uk/CSTR/GlobalPhone>, although only for a few utterances

partitioning can be replicated by other future works. Tab. 3.1 shows the total data amounts for each language and partition.

Notice that the data amount varies across languages for a given partition (i.e. languages are represented unequally in the corpus – compare Wu and Japanese), and even the declared 80%-10%-10% split from the original GlobalPhone partitioning is not too reliable in some cases – for instance Thai, German, Mandarin and Spanish – although, in other cases, it does result in a split close to the declared one (such as for Korean, Swedish and Turkish). Further, the per-speaker amounts of data are clearly not equal, see Wu where there are 6 training-set speakers vs 4 testing-set speakers, yet the testing set has a greater total amount of data than the training set. While attempting at a perfect split is possible, instead, I appreciate the realisticity of certain languages being under-represented, and account for this in training the x-vector classifier (see [TO-DO: add reference once the section is written] for details of how this is implemented).

	TRAINING	ENROLLMENT	EVALUATION	TESTING
AR	14.5	1.4	1.7	1.4
BG	15.1	2.0	2.3	2.0
CH	23.7	3.1	2.0	2.4
CR	10.5	1.6	2.0	1.8
CZ	23.7	3.1	2.4	2.7
FR	20.1	2.7	2.1	2.0
GE	13.2	1.7	2.0	1.5
JA	29.2	3.0	0.9	0.8
KO	14.6	2.2	2.2	2.1
PL	17.0	2.3	2.8	2.3
PO	20.8	1.9	1.6	1.8
RU	17.3	2.5	2.5	2.4
SP	16.1	2.3	2.1	1.7
SW	15.4	2.0	2.1	2.2
TA	10.7	2.2	1.1	1.3
TH	22.0	2.9	2.3	0.9
TU	11.6	1.7	2.0	1.9
VN	15.8	1.7	1.4	0.8
WU	1.0	0.8	0.7	1.1

Table 3.1: Amount of data (in hours) for each language and partition after excluding invalid data

	TRAIN	ENROLLMENT	EVALUATION	TESTING
AR	57	017, 023, 056, 104, 105, 114, 135	014, 030, 047, 067, 109, 115, 137	011, 012, 027, 050, 051, 055, 066
BG	56	042, 048, 056, 065, 088, 104, 107	051, 055, 058, 084, 090, 100, 106	040, 059, 063, 068, 095, 109, 110
CH	98	004, 005, 012, 034, 035, 036, 045, 048, 052, 053, 057, 120, 126	028, 029, 030, 031, 032, 039, 040, 041, 042, 043, 044	080, 081, 082, 083, 084, 085, 086, 087, 088, 089
CR	63	001, 010, 018, 026, 029, 032, 074, 092, 093	033, 034, 035, 036, 046, 048, 051, 053, 054, 057	037, 038, 039, 040, 041, 042, 043, 044, 045, 047

CZ	72	005, 009, 031, 033, 034, 036, 053, 061, 071, 072	083, 085, 087, 089, 091, 093, 095, 097, 099, 101	084, 086, 088, 090, 092, 094, 096, 098, 100, 102
FR	74	009, 028, 030, 031, 033, 052, 064, 065, 070, 076	082, 083, 084, 085, 086, 087, 088, 089	091, 092, 093, 094, 095, 096, 097, 098
GE	58	015, 017, 046, 056, 058, 065, 074	001, 002, 003, 004, 008, 010	018, 020, 021, 026, 029, 073
JA	119	002, 021, 037, 048, 051, 053, 057, 064, 086, 201, 202, 206, 209, 221	009, 031, 046, 081, 091	006, 025, 045, 047, 088, 101
KO	70	018, 030, 043, 044, 057, 065, 074, 078, 079, 090	006, 012, 025, 040, 045, 061, 084, 086, 091, 098	019, 029, 032, 042, 051, 064, 069, 080, 082, 088
PL	70	008, 018, 032, 076, 083, 085, 088, 096, 099	005, 011, 012, 030, 040, 041, 046, 063, 090, 097	001, 004, 009, 023, 031, 033, 043, 044, 050, 098
PO	75	070, 071, 106, 109, 117, 120, 128, 146, 148, 150	064, 072, 102, 103, 104, 132, 133, 134	135, 137, 138, 139, 142, 143, 312
RU	84	017, 023, 035, 041, 088, 101, 114, 115, 119, 121, 123	005, 033, 042, 065, 078, 097, 103, 106, 110, 122	002, 027, 036, 063, 069, 092, 102, 104, 109, 112
SP	72	037, 041, 048, 059, 060, 064, 074, 083, 098, 099	001, 002, 003, 004, 005, 006, 007, 008, 009, 010	011, 012, 013, 014, 015, 016, 017, 018
SW	70	001, 013, 023, 070, 072, 077, 078, 080, 084	045, 046, 047, 048, 049, 066, 067, 068, 069	040, 041, 042, 043, 044, 060, 061, 062, 063, 064
TA	35	022, 028, 038, 047	003, 030, 035, 039	001, 005, 033, 037
TH	73	001, 005, 027, 031, 041, 065, 067, 077, 083	023, 025, 028, 037, 045, 061, 073, 085	101, 102, 103, 104, 105, 106, 107, 108
TU	69	010, 017, 044, 048, 049, 051, 064, 091, 093, 096	001, 002, 003, 005, 006, 008, 013, 014, 015, 016, 019	025, 030, 031, 032, 037, 039, 041, 046, 056, 063
VN	103	012, 029, 031, 036, 043, 050, 053, 070, 077, 118, 119, 128	200, 201, 202, 203, 204, 205, 206, 207, 208	102, 103, 106, 110, 113
WU	6	008, 011, 013, 015	002, 004, 006, 017	001, 003, 005, 009

Table 3.2: The partitioning of each language’s speakers into the 4 partitions. For enrollment, evaluation and testing sets, speaker IDs are shown. For the training set, the rest of the speakers was used and only the number of speakers is shown.



# Chapter 4

## Implementation

In this chapter, I describe the entire architecture and computing environment. Importantly, for the early implementation stage, I teamed up with Paul Moore and a big part of the baseline system was implemented jointly – with his contributions. Naturally, in my writing, I focus more on parts which I developed independently, without Paul’s help. These are in particular: splitting utterances into shorter segments; early and late fusion of different feature types; choosing and adapting the classifier; exploring the number of training epochs of the TDNN.

As a matter of fact, the codebase of this project consists primarily of adapted existing implementations. Nevertheless, I describe the architecture in detail. However, I make it explicit where the design or hyper-parameter decisions were made by myself (possibly in collaboration with Paul). In the rest of the cases, the reader should implicitly assume that the decisions were made by the authors of the original implementations and found to work well. Importantly, my work does not primarily focus on optimizing the architecture or hyperparameters – only on the different ways in which various feature types can be used and combined to improve LID performance.

### 4.1 The Kaldi toolkit

As hinted in earlier chapters, the system was implemented in the Kaldi toolkit<sup>1</sup>. First presented by its authors in (Povey et al., 2011) around the time i-vectors were first proposed, Kaldi is the most popular research toolkit for speech processing – originally developed for ASR, but nowadays also used for speaker and language recognition. As an actively maintained open-source toolkit, it reflects the state of the art research and contains working implementations of many accompanying published papers. (After all, the top-class research community significantly overlaps with the community of Kaldi developers.)

To understand what it means to implement an architecture like the one I use in Kaldi, one should understand and appreciate the three layers of code Kaldi contains:

---

<sup>1</sup><http://kaldi-asr.org/>

1. **Kaldi binaries** (written mostly in C++) provide low-level, task-specific functions, e.g. for computing MFCC features, training a generic neural network or computing VAD. The binaries typically require input and output in the specific Kaldi formats<sup>2</sup> and can be heavily customised by providing values for numerous parameters. Multiple binaries are often used sequentially to create a multi-step pipelines.
2. **Kaldi scripts** (written mostly in Bash) provide convenient implementations of more high-level steps combining multiple binaries, as well as code for data manipulation. In many cases, these scripts come with useful default values (for the binaries' parameters) empirically shown to work well.
3. **Kaldi recipes** are end-to-end implementations utilising the code from layers 1 and 2 and corresponding to particular studies or experiments, or just replicating a study with a different dataset. They typically come with one top-level script called `run.sh` which calls all other recipe-specific code (or code from layers 1 and 2) and can be run to replicate the study in question provided that one has the appropriate datasets available.

## 4.2 Kaldi recipes used in this work

My entire implementation is simply a new Kaldi *recipe* (see previous section) and – by the nature of Kaldi – it re-uses a lot of existing code, including the following Kaldi recipes:

- The GlobalPhone (GP) recipe<sup>3</sup>, corresponding to Lu et al. (2012), inspired my implementation of preprocessing the GlobalPhone corpus and connecting it with the rest of the architecture – however, with almost all the final code being heavily adapted or re-written altogether with respect to the recipe (see Section 3.3).
- The Speaker Recognition Evaluation 2016 (SRE16) recipe<sup>4</sup>, corresponding to Snyder et al. (2018b) contains an end-to-end speaker verification pipeline using x-vectors. This recipe forms the basis for my implementation because there is no published recipe corresponding to the LID x-vector paper (Snyder et al., 2018a), but the two papers by Snyder et al. use the very same x-vector architecture, only substantially differing in the choice of the subsequent classifiers. Fortunately, the LID x-vector paper provides enough information on aspects that are different SRE x-vector architecture, which makes it possible to replicate the x-vector LID system very closely.
- The LRE07 recipe<sup>5</sup>, corresponding to Sarma et al. (2018), provided me with scripts for training a logistic regression classifier for LID, which I adapted and connected with the x-vector system.

---

<sup>2</sup>for more information on Kaldi I/O see <http://kaldi-asr.org/doc/io.html>

<sup>3</sup><https://github.com/kaldi-asr/kaldi/tree/master/egs/gp/s5>

<sup>4</sup><https://github.com/kaldi-asr/kaldi/tree/master/egs/sre16/v2>

<sup>5</sup><https://github.com/kaldi-asr/kaldi/tree/master/egs/lre07/v2>

Apart from the time spent on familiarising myself with Kaldi and understanding, adapting and integrating the enumerated recipes, I put the most effort into connecting the corpus with the x-vector architecture, implementing early and late fusion, computing the various feature types, and building a robust end-to-end pipeline (within the `run.sh` script) which could be run without any changes either as a whole or step by step, only changing a simple configuration file to switch between different experiments.

### 4.3 Choice of classifier

As mentioned earlier, although the SRE16 Kaldi recipe provides a complete implementation of the x-vector TDNN architecture, it does not use the same classifier as the LID x-vector paper by Snyder et al.. Instead, it uses the Probabilistic Linear Discriminant Analysis (PLDA) model for *binary* classification (because the task is speaker *verification*). The LID x-vector paper, on the other hand, uses a multi-class classifier: more precisely, a GMM classifier trained using the Maximum Mutual information (MMI) criterion as presented by McCree (2014). Unfortunately, as I found out and confirmed with David Snyder, this classifier is not implemented in Kaldi (as of January 2019). While implementing the classifier by myself was an option, it would have consumed too much time and effort – whether implemented in Kaldi (including substantial changes to the Kaldi binaries), or separately (outside the toolkit). Bear in mind that my aim is to build *some* well-performing architecture that leverages the power of x-vectors and enables me to experiment with different feature types and fusion strategies. Hence, I decided to use a different classifier – ideally more established and implemented in Kaldi.

Recently, various types of classifiers have been used on top of i-vector and d-vector models, without a clear comparison – suggesting that the choice of classifier is not nearly as important as back end producing utterance-level vectors. While various varieties of Gaussian classifiers are popular (González et al., 2013; McCree, 2014; Plchot et al., 2016), other approaches such as Support vector machines (Tkachenko et al., 2016; Martinez et al., 2011) and logistic regression (Sarma et al., 2018; Martinez et al., 2011, 2012). I decided to use multi-class logistic regression as it is already implemented in Kaldi in the context of LID as part of the LRE07 recipe. Consulting my decision with David Snyder (2018) only assured me that this was a solid choice.

Of course, there are downsides to using this classifier. As McCree (2014) points out, unlike the discriminative logistic regression, a generative classifier (in this case GMM) can provide out-of-set rejection decisions. This, however, does not speak against logistic regression in my particular case as I do *closed-set* classification. Perhaps more serious inherent disadvantage is that logistic regression uses only linear decision boundaries – compare with GMMs with full covariance matrices, which can model more complex quadratic boundaries. The way this issue is addressed in Kaldi’s implementation is by making logistic regression a mixture model, i.e. modelling each class by multiple decision boundaries (for more details see Section 4.4). This way, more complex boundaries can be modelled in the observation space. Because training GMMs with full covariance matrices is also known to be data demanding and computationally

expensive, logistic regression might potentially be preferred where the classifier is to be trained on limited enrollment data: I refer to the findings of Snyder et al. that solid LID performance can be achieved even for languages unseen by the x-vector TDNN. This scenario, however, is not very relevant for LID systems which are parts of ASR systems – there, just training acoustic models for ASR requires amounts of data that would likely be sufficient even for a simple GMM with a full covariance matrix.

## 4.4 Final architecture

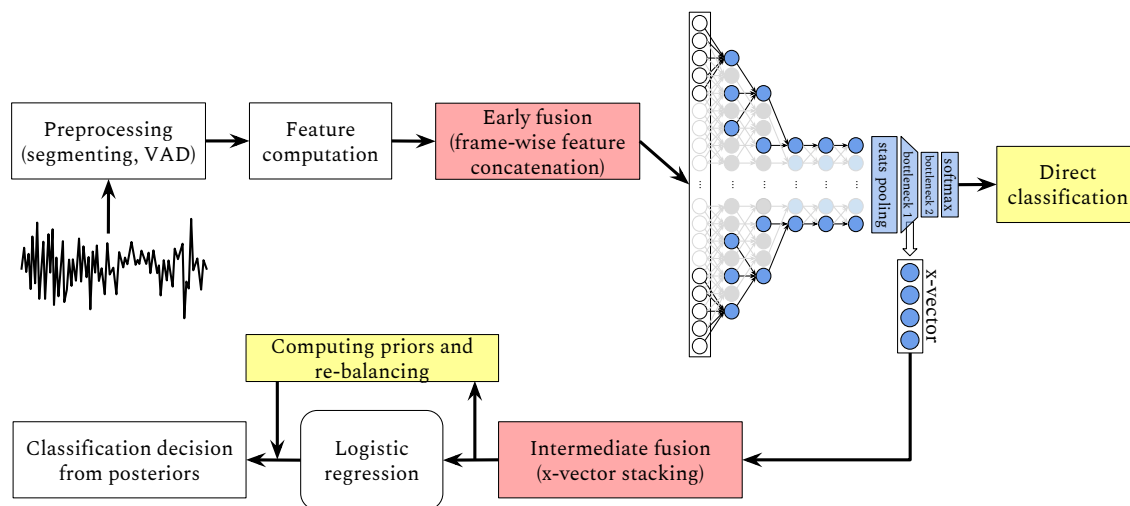


Figure 4.1: An end-to-end overview of my system. Red marks the fusion steps which can be skipped, yellow marks steps only performed at training time.

mention that implementing this end-to-end thing took the most time and running experiments was really fast

pipeline overview (diagrams showing training and evaluation pipelines together as a whole) - feature computation - concatenating features (early fusion) - removing silence (VAD) - producing x-vector - concatenating with another x-vector (late fusion) - classifying

data preparation feature computation - existing scripts re-used for computing MFCCs (make\_mfcc.sh), I adapted the code for adding SDCs and deltas to MFCCs - existing scripts re-used for adding pitch - energy feature computed simply as a special case of MFCC: taking only 0th coefficient and using raw log energy instead of it

early fusion - I wrote script for frame-wise concatenation - done for all data partitions alike - 3s mean normalisation before x-vector extraction and also for tdnn training samples – for all feature types

training the tdnn - architecture as described in Section 2.4, no changes to the code from the recipe - corresponds to that described in Snyder et al. (2018a): ReLU nonlinearities, 64-sample minibatches, NG-SGD training with EFFECTIVE lr exponentially decreasing from 0.001 to 0.0001, described in Povey et al. (2014) (also: Trained



with 3 parallel jobs, parameter shrinking and with momentum with the momentum-term coefficient  $\mu = 0.5$  described therein), dropout scheme is 0 (at 0%-20%), 0.1 (at 50%), 0 (at 100%). batch normalisation. maximum parameter change: The maximum change in parameters allowed per minibatch, measured in Euclidean norm over the entire model (change will be clipped to this value): 2.0. Backstitch is NOT used. - utts ;5s (500 frames) removed from training data - training data repeated 35 times, distributed into archives of roughly 50000000 frames each (around 140h). 70 training archives - number of epochs translated into a number of iterations (each iteration = one archive) - samples globally shuffled before training, then shuffled each iteration within batches of 1000 - training takes around 19 hours (see Section 4.6) - uses the nnet3 Kaldi implementation - 3 initial and 3 final jobs (i.e. simply 3 parallel jobs throughout training) - number of parameters: 4.6M (MFCC+deltas+pitch+energy) to 4.4M (energy) - optimising multi-class cross-entropy

extracting x-vectors - done using the sre16 code without any changes - ignoring utts ;1s - 512-D x-vectors - takes around 1.5h for enroll, eval and test data combined (when using 32 parallel jobs) - have: 17212 30s enrollment segments + 20349 10s eval segments + 18710 10s test segments =, around 100ms per segment (to produce x-vector)

late fusion - I wrote script for concatenating 2 or 3 x-vectors together for each utterance, resulting in 1024-D or 1536-D x-vectors

training the classifier - all vectors length-normalised before fed into classifier! - whitening or dimensionality reduction is not performed - rebalancing priors:  $(count(lang - test)/count(lang - train))^{(prior - scale)}$  where prior-scale=0.7 (same as in the original LRE07 recipe, and generally recommended values are between 0.5 and 1.0) - add graph showing distribution of priors and eval utt counts - optimising cross-entropy - hyperparameters: - max-steps: Maximum steps in L-BFGS (default: 20). - normalizer: Coefficient for L2 regularization (default: 0.0025). - power: Power rule for determining the number of mixtures to create (default: 0.15). - mix-up: Target number of mixture components to create (default: 0). - first trained for max-steps with number of classes = K (19) - then split each class into a number of "sub-classes" like a mixture model. done iteratively, in each iteration splitting the biggest cluster into two. what 'biggest' means is determined by the 'power' parameter:  $(total - occ^power)/num - components$  is taken as the adjusted per-component occupancy of the cluster, thus using using power closer to 0 means that the number of components will be more equal across all classes, whereas for big power (close to 1) big classes will be split into many more components than smaller classes - then trained for another max-steps but with number of classes corresponding to mix-up. the decision boundaries are initialised from the K decision boundaries with some added random noise so they would differ slightly. essentially, each class is modelled using multiple decision boundaries but all boundaries (mixture components) have the same weight - takes around 3.5 minutes - inference on test set, together with computing classification results, takes around 4s (single-threaded). 18663 10s test utts, hence around 0.2ms per utterance (very solid!) using one CPU

## 4.5 Computing environment

the School of Informatics student GPU cluster NVIDIA GTX1060 6GB CPUs were: Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz Slurm job scheduler using one GPU per parallel job (i.e. 3 GPUs in TDNN training)

## 4.6 Hyperparameters

decided: TDNN layers, activation function, learning algorithm, ... decided for me mainly in the SRE16 recipe and in the X-vector paper

tuned (using the baseline setting, i.e. vanilla MFCCs, evaluation using 10s eval segments, optimising C-primary, see next chapter): number of TDNN training epochs  $E$ , logit hyperparameters -  $E$  was tuned to provide a good tradeoff between training time and performance

# Chapter 5

## Experiments

Intro: I will compare a selection of acoustic and prosodic features. Despite their reported potential, I don't use BNFs in this work, as they are basically just a higher-level feature based on the acoustic MFCC information (at least the BNFs in Snyder et al. (2018a)).

Evaluation metric:  $C_{primary}$ , consistent with NIST LREs. Elaborate a bit more on the meaning of the metric, maybe compare with accuracy and other simpler metrics.

### 5.1 Baseline

vanilla MFCCs (no deltas): also comment on the decisions made regarding MFCC MFCC configuration

$\leq 30s$  enrollment segments,  $\leq 10s$  eval/test segments (should be possible to also report exact average segment length for each set)

### 5.2 Shifted delta cepstra

Want to see whether context aggregation in the form of added deltas in SDCs (compared to vanilla MFCCs) can improve performance, since the TDNN does context aggregation of its own.

### 5.3 KaldiPitch+Energy vectors

Calculating pitch even for unvoiced frames using the algorithm presented by Ghahremani et al. (2014). Adding raw energy to model stress. Extremely low-dimensionality feature vectors, but will see how the TDNN and classifier trained on these can do

prosodic LID. Hoping to achieve some sensible results, probably much worse than with MFCCs or SDCs.

## 5.4 MFCC/SDC + KaldiPitch+Energy

Taking the winner from MFCC/SDC, and concatenating those features with Kaldi pitch and raw energy values. Training the TDNN and classifier on that. Hopefully, seeing an improvement.

## 5.5 Fusion of MFCC/SDC and KaldiPitch+Energy scores

Stretch goal, likely to be delayed for Year 5 (or abandoned if there are more attractive ideas)

Same as previous section, but scores computed separately (using two systems, one trained on acoustic features, the other one on prosodic) and fused using a logit fuser. Probably using evaluation set for training the fuser (although, ideally, a separate data portion would be reserved for that).

## 5.6 Timeline for the experiments

1. Finished: System using MFCC and SDC.
2. By January 28th: Choose the TDNN and logit hyperparameters. TDNN using MFCCs has already been trained for, 2, 3, ... 10 epochs, now need to 1) establish reasonable logit parameters (driven by evaluation-set performance on x-vectors from TDNN trained for 3 epochs), 2) use that logit config to score the TDNNs with different number of training epochs, 3) choose a number of epochs that is a reasonable compromise between performance and training runtime
3. By February 4th: Have MFCC vs SDC comparison carried out (includes baseline results using MFCCs). Have KaldiPitch and raw energy features implemented (both are straightforward to compute with Kaldi – the energy is just computing MFCCs with raw log energy instead of C0, and discarding all but the energy-corresponding resulting coefficient). Have splicing of MFCC/SDC features with prosodic features implemented.
4. By February 11th: Have results of KaldiPitch+Energy experiments and of the acoustic+prosodic experiment (MFCC/SDC + KaldiPitch+Energy)

# Chapter 6

## Results

Reporting: overall  $C_{primary}$ , accuracy (for illustrative purposes), confusion matrix (to see which language pairs are confusing)

Focus on Slavic languages, since there is so many of them (Czech, Croatian, Polish, Russian, Bulgarian) and intonation can be very characteristic and important here (my own intuition, based on my knowledge of Slavic languages).



# **Chapter 7**

## **Discussion**





# **Chapter 8**

## **Future work**

### **8.1 Plans for Part 2 of the MInf project**

### **8.2 Other future ideas**

Everything that is sensible but unlikely in Year 5.



# **Chapter 9**

## **Conclusions**



# Bibliography

- Boersma, P. and Weenink, D. (2019). Praat: doing phonetics by computer [computer program]. version 6.0.49. <http://www.praat.org/>.
- Crystal, D. (2008). *A Dictionary of Linguistics and Phonetics*. Blackwell, 6<sup>th</sup> edition.
- Dehak, N., Kenny, P., Dehak, R., Dumouchel, P., and Ouellet, P. (2011). Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, 19:788–798.
- Ferrer, L., Lei, Y., McLaren, M., and Scheffer, N. (2016). Study of senone-based deep neural network approaches for spoken language recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24:105–116.
- Ghahremani, P., BabaAli, B., Povey, D., Riedhammer, K., Trmal, J., and Khudanpur, S. (2014). A pitch extraction algorithm tuned for automatic speech recognition. *2014 IEEE ICASSP*, pages 2494–2498.
- Goedemans, R. and van der Hulst, H. (2013a). Fixed stress locations. In Dryer, M. S. and Haspelmath, M., editors, *The World Atlas of Language Structures Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.
- Goedemans, R. and van der Hulst, H. (2013b). Weight-sensitive stress. In Dryer, M. S. and Haspelmath, M., editors, *The World Atlas of Language Structures Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.
- González, D. M., Lleida, E., Ortega, A., and Miguel, A. (2013). Prosodic features and formant modeling for an ivector-based language recognition system. *2013 IEEE ICASSP*, pages 6847–6851.
- Grabe, E. (2004). Intonational variation in urban dialects of English spoken in the British Isles. *Regional variation in intonation*, pages 9–31.
- Holmes, J. and Holmes, W. (2002). *Speech Synthesis and Recognition*. Taylor & Francis, Inc., Bristol, PA, USA, 2<sup>nd</sup> edition.
- Lin, C.-Y. and Wang, H.-C. (2005). Language identification using pitch contour information. *Proc. 2005 IEEE ICASSP*, 1:I/601–I/604.
- Lu, L., Ghoshal, A., and Renals, S. (2012). Maximum a posteriori adaptation of subspace Gaussian mixture models for cross-lingual speech recognition. In *Proc. 2012 IEEE ICASSP*, pages 4877–4880. IEEE.

- Mak, M.-W. and Chien, J.-T. (2016). Interspeech 2016 tutorial: Machine learning for speaker recognition. [http://www1.icsi.berkeley.edu/Speech/presentations/AFRL\\_ICSI\\_visit2\\_JFA\\_tutorial\\_icsitalk.pdf](http://www1.icsi.berkeley.edu/Speech/presentations/AFRL_ICSI_visit2_JFA_tutorial_icsitalk.pdf).
- Martinez, D., Burget, L., Ferrer, L., and Scheffer, N. (2012). ivector-based prosodic system for language identification. In *Proc. 2012 IEEE ICASSP*, pages 4861–4864. IEEE.
- Martinez, D., Plhot, O., Burget, L., Glembek, O., and Matějka, P. (2011). Language recognition in iVectors space. In *Proc. Interspeech 2011*.
- McCree, A. (2014). Multiclass discriminative training of i-vector language recognition. In *Proc. Odyssey 2014*, pages 166–172.
- Metze, F., Sheikh, Z. A. W., Waibel, A., Gehring, J., Kilgour, K., Nguyen, Q. B., and Nguyen, V. H. (2013). Models of tone for tonal and non-tonal languages. In *Proc. 2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 261–266.
- Paul, D. B. and Baker, J. M. (1992). The design for the Wall Street Journal-based CSR corpus. In *Proceedings of the workshop on Speech and Natural Language*, pages 357–362. Association for Computational Linguistics.
- Plhot, O., Matejka, P., Fér, R., Glembek, O., Novotný, O., Pešán, J., Veselý, K., Ondel, L., Karafiát, M., Grézl, F., et al. (2016). BAT system description for NIST LRE 2015. In *Proc. Odyssey 2016*.
- Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G., and Vesely, K. (2011). The Kaldi speech recognition toolkit. In *Proc. IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society.
- Povey, D., Zhang, X., and Khudanpur, S. (2014). Parallel training of DNNs with natural gradient and parameter averaging. *arXiv*, abs/1410.7455.
- Prieto, P. and Roseano, P. (2018). *Prosody: Stress, rhythm, and intonation*, pages 211–236. Cambridge University Press. Retrieved from [https://www.researchgate.net/publication/329541457\\_Prosody\\_Stress\\_rhythm\\_and\\_intonation](https://www.researchgate.net/publication/329541457_Prosody_Stress_rhythm_and_intonation).
- Sarma, M., Sarma, K. K., and Goel, N. K. (2018). Language recognition using time delay deep neural network. *arXiv*, abs/1804.05000.
- Schultz, T. (2002). Globalphone: a multilingual speech and text database developed at Karlsruhe University. In *Proc. ICSLP 2002*, pages 345–348.
- Schultz, T., Vu, N. T., and Schlippe, T. (2013). GlobalPhone: A multilingual text & speech database in 20 languages. *Proc. 2013 IEEE ICASSP*, pages 8126–8130.
- Shimodaira, H. and Renals, S. (January 17, 2019). Speech signal analysis: Asr lectures 2&3. <http://www.inf.ed.ac.uk/teaching/courses/asr/2018-19/asr02-signal-handout.pdf>.

- Snyder, D. (December 24, 2018). Language identification with x-vectors: Choice of classifier [online forum comment]. <https://groups.google.com/forum/#!topic/kaldi-help/v6Uh7avv-cY>.
- Snyder, D., Garcia-Romero, D., McCree, A., Sell, G., Povey, D., and Khudanpur, S. (2018a). Spoken language recognition using x-vectors. In *Proc. Odyssey 2018*, pages 105–111.
- Snyder, D., Garcia-Romero, D., Sell, G., Povey, D., and Khudanpur, S. (2018b). X-vectors: Robust DNN embeddings for speaker recognition. *2018 IEEE ICASSP*, pages 5329–5333.
- Tkachenko, M., Yamshinin, A., Lyubimov, N., Kotov, M., and Nastasenko, M. (2016). Language identification using time delay neural network d-vector on short utterances. In *Proc. SPECOM*, pages 443–449. Springer.
- Torres-Carrasquillo, P. A., Singer, E., Kohler, M. A., Greene, R. J., Reynolds, D. A., and Deller, J. R. (2002). Approaches to language identification using Gaussian mixture models and shifted delta cepstral features. In *Proc. Interspeech 2002*.
- Variani, E., Lei, X., McDermott, E., Moreno, I. L., and Gonzalez-Dominguez, J. (2014). Deep neural networks for small footprint text-dependent speaker verification. pages 4052–4056.
- Zheng, F. and Zhang, G. (2000). Integrating the energy information into MFCC. In *Proc. ICSLP 2000*.