
# 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

# 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 identificaion* (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.

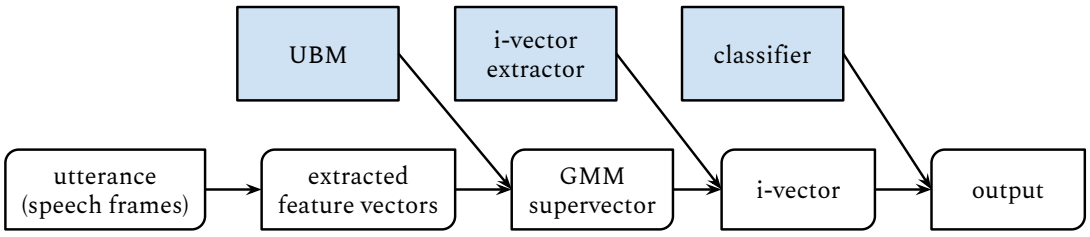


Figure 2.1: Language identification using a typical i-vector system.

The main components of a typical i-vector system, together with their use for prediction, are shown in Figure 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:

$$M_X = m + Tw_X \tag{2.1}$$

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 $0^{\text{th}}$, $1^{\text{st}}$ and $2^{\text{nd}}$ 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. Figure 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 Table 2.1 (notice the difference between the interval and set notation); "layer context" meant relative to the preceding layer.

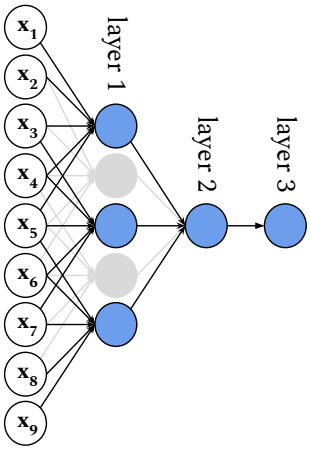


Figure 2.2: Example of a time-delay neural network.

| | 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 Figure 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 Figure 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.

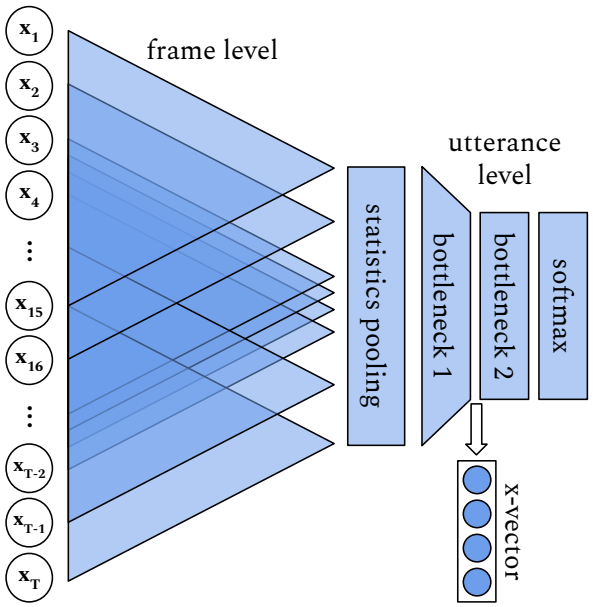


Figure 2.3: A high-level sketch of the x-vector TDNN from Snyder et al..

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 lightly 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

### 2.5.1 Acoustic features

### 2.5.2 Prosodic features

# Chapter 3

# Data

Intro: Historically, speech corpora were meant for ASR, but now used for LID and SID as well. Since 1996, NIST has also been organising Language Recognition Challenges (LREs), and the corpora used for LREs are now the typically used ones when it comes to evaluating different systems. However, NIST traditionally focuses on telephone, narrowband (8kHz) speech, which represents only one part of all the possible settings in which LID or SID are deployed. In this work, I use a relatively small (¡ 40GB) corpus which, however, has many qualities that the various NIST corpora lack.

## 3.1 GlobalPhone

Presented by Schultz et al. (2013), originally for ASR. Wideband (16kHz), recorded native speakers of 23 languages reading newspaper articles – very similar to the English corpora such as CSR-I, which are based on Wall Street Journal articles. Unlike in NIST corpora, the recording equipment and conditions vary very little. One disadvantage of GlobalPhone is the lack of spontaneous speech. On the other hand, the language goes well beyond the simple conversational language found in the NIST telephone speech corpora.

## 3.2 Data partitioning

GlobalPhone comes with a partitioning of each language's data into training, development and evaluation datasets, with the sizes of the datasets being roughly 80%, 10%, 10%, respectively. However, for the purposes of the x-vector architecture, 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 ('development' in the GlobalPhone terminology) – for tuning the hyperparameters of both the TDNN and the classifier,

4. testing set – for final performance evaluation on unseen data.

In order to use GlobalPhone with the x-vector system, I allocated part of the training data for enrollment. I tried to preserve the GlobalPhone development sets and evaluation sets (which I refer to as evaluation and testing sets, respectively), in order to enable fair comparison of my results obtained on those data portions with any other works that use the GlobalPhone corpus.

Taking into account the relatively small number of parameters of the x-vector classifier when compared to the x-vector TDNN, I split the training data such that approximately only one 8th is used for enrollment. This way, the training data accounts for roughly 70% of the whole corpus, while the 3 other portions are roughly 10% each.

Importantly, GlobalPhone (as of the GlobalPhone Documentation v4.1) still misses the partitioning for certain languages:

1. no partitioning for Czech, Polish, Tamil, Swahili, Ukrainian, Vietnamese and Shanghai Chinese,

2. only partial partitioning for Arabic, French, Japanese, Russian.

The GlobalPhone Kaldi recipe contains an extended partitioning, which fixes Czech, French, Japanese, Polish, Russian, Swahili and Vietnamese. For the rest of the languages with incomplete partitioning (Arabic, Tamil, Ukrainian and Shanghai Chinese), I created the partitioning myself, following the same approach as the GlobalPhone authors: "No speaker appears in more than one group and no article was read by two speakers from different groups" (TO-DO: reference the GP docs).

Some languages didn't have speaker-article data; for those, the partitioning was done randomly.

For some languages, I could not construct a partitioning with zero article overlap; for those, I tried to at least minimise the overlap.

## 3.3 Initial data preprocessing

Basically, SHN to WAV

Splitting long utterances into shorter ones uniformly, to be able to do classifier training and end-to-end evaluation on segments of different lengths – like in Snyder et al. (2018a). One future improvement would be to not do uniform splitting, but rather split on breaks – to prevent potentially bad edge effects.

## 3.4 Invalid data

This was discovered while preprocessing the data from .shn to .wav:

1. Hausa, Swahili and Ukrainian have broken data.
2. Bulgarian, German, Russian, Turkish and Vietnamese have one broken utterance recording each – not a big deal, as the number of utterances per language is in hundreds.
3. Portuguese has 2 speakers with almost all data broken (these were discarded), and other 10 speakers with up to 3 broken utterance recordings each (these were keps)

This leaves us with 19 languages, which are used further in this work: Arabic (AR), Bulgarian (BG), Mandarin Chinese (CH), Croatian (CR), Czech (CZ), French (FR), German (GE), Japanese (JA), Korean (KO), Polish (PL), Portuguese (PO), Russian (RU), Spanish (SP), Swedish (SW), Tamil (TA), Thai (TH), Turkish (TU), Vietnamese (VN) and Shanghai Chinese (WU).

# Chapter 4

# Implementation

## 4.1 The Kaldi toolkit

System was built in Kaldi (Povey et al., 2011). (Introduce Kaldi.)

## 4.2 Adapted implementations

Describe the SRE16 Kaldi recipe, the GlobalPhone ASR recipe, and how they were combined (also using the information from Snyder et al. (2018a)) to reproduce the LID x-vector system.

## 4.3 Choice of classifier

The SRE16 recipe uses PLDA, but for verification, not for classification. For our purposes, we needed a proper classifier. Current popular and well-performing classifiers are various flavours of GMM and logistic regression, with no clear winner. Snyder et al. (2018a), for instance, used GMM trained using MMI – based on McCree (2014). I decided to re-use a model which is already implemented in the LRE07/v2 Kaldi recipe – logistic regression. The decision was also consulted with Snyder (r 24). One theoretical downside of logistic regression is that the resulting decision boundaries are linear, whereas with GMM (trained with full co-variance matrix), one can achieve more complex quadratic decision boundaries. On the other hand though, training a full covariance requires much data, and the enrollment set would likely be not big enough. Additionally, the Kaldi logit can describe each class using more than linear boundary (see next section), so the decision boundaries should be complex enough to be able to model the observed data well.

## 4.4 Final architecture

Description of x-vector+GP architecture (highlighting own contributions)

how the whole pipeline works: go into much more detail than in the Related work section

description of the Kaldi logit, which models each class using multiple ”mixtures”, i.e. multiple boundaries

mention possibility of direct classification with TDNN and that I focus on using separate classifier because it provides extra flexibility and because it was shown to perform slightly better

## 4.5 Computing environment

cluster, Slurm, GPUs, parallelisation, rough runtimes of the different stages (TDNN training, x-vector extraction, logit training, inference)

## 4.6 Hyperparameters

decided: TDNN layers, activation function, learning algorithm (TO-DO: read about shrinking)

tuned (using the baseline setting, see next chapter): number of TDNN training epochs, logit hyperparameters

# 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


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.

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.

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.

Lin, C.-Y. and Wang, H.-C. (2005). Language identification using pitch contour information. *Proc. 2005 IEEE ICASSP*, 1:I/601–I/604 Vol. 1.

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`.

Martinez, D., Plchot, O., Burget, L., Glembek, O., and Matějka, P. (2011). Language recognition in iVectors space. In *Interspeech 2011*.

McCree, A. (2014). Multiclass discriminative training of i-vector language recognition. In *Proc. Odyssey 2014*, pages 166–172.

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 *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society.

Schultz, T., Vu, N. T., and Schlippe, T. (2013). GlobalPhone: A multilingual text & speech database in 20 languages. *2013 IEEE ICASSP*, pages 8126–8130.

Snyder, D. (2018, December 24). 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 *SPECOM*, pages 443–449. Springer.

Variani, E., Lei, X., McDermott, E., Moreno, I. L., and Gonzalez-Dominguez, J. (2014). Deep neural networks for small footprint text-dependent speaker verification. In *2014 IEEE ICASSP*, pages 4052–4056. IEEE.