NTNU – Trondheim
Norwegian University of
Science and Technology

iVector Based Language Recognition

Åsmund Einar Haugland Tokheim

Master of Science in Communication Technology
Submission date: June 2012
Supervisor: Torbjørn Svendsen, IET
Co-supervisor: Mehdi Soufifar, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Abstract

The focus of this thesis is an fairly new approach to phonotactic language recognition, i.e. identifying a language from the sounds in an spoken utterance, known as iVector subspace modeling. The goal of the iVector is to compactly represent the discriminative information in a utterance so that further processing of the utterance is less computationally intensive. This might enable the system to be trained with more data, and thereby reach an higher performance. We present both the theory behind iVectors and experiments to better fit the iVector space to our development data. The final system got comparable result to our baseline PRLM system on the NIST LRE03 30 second evaluation set.

Sammendrag

Et automatisk språkidentifiseringssystem er et program som gjennkjenner språket som ble brukt i tale. I denne oppgaven redgjør vi for en tilnærming til et slikt system som bruker en såkalt iVektor underromsrepresentasjon av tale. Med denne representasjonen forkastes informasjon som ikke er nyttig for å gjenkjenne språket. Dette kan føre til raskere behandling av talen, noe som lar oss trene systemet med mer data og dermed oppnå høyere ytelse. Vi har også gjort forsøk på å få underrommet bedre tilpasset til data den ikke er trent fra. Det resulterende systemet oppnådde tilsvarende ytelse som mer tradisjonelle språkidentifiseringssystemer på segmenter med 30 sekunder tale fra NIST LRE03 testsettet.

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) for partial fulfillment of the requirements for the degree of Master of Science. The work has been performed at the Department of Electronics and Telecommunications in the spring semester of 2012.

I would like to thank my supervisor Professor Torbjørn Svendsen for sharing his valuable insight in language recognition systems, and my co-supervisor Mehdi Soufifar for sharing his expertise in iVector systems.

Contents

Abstract i

Sammendrag iii

Preface v

Contents vii

1 Introduction 1
- 1.1 Problem Description . . . 1
- 1.2 Objectives for the Thesis . . . 2
- 1.3 Structure . . . 2

I Theory 5

2 Language Recognition Systems 7
- 2.1 Language Detection vs. Identification . . . 7
- 2.2 Evaluation Metrics . . . 8
- 2.3 Language Characterizations . . . 8
- 2.4 System Overview . . . 9
- 2.5 Phoneme Recognizers . . . 10
- 2.6 Language Models and Classifiers . . . 12
- 2.7 Backend Calibration of Score Vectors . . . 12

3 Statistical Language Modeling 15
- 3.1 Model training . . . 15
- 3.2 Model Smoothing . . . 16

4 iVector Subspace Modeling 19
- 4.1 Background . . . 19
- 4.2 Interpretation of iVector Model . . . 20
- 4.3 Model Training . . . 20
- 4.4 Extraction of iVectors from the Model . . . 22
- 4.5 The Iterative Update Process . . . 22

5 Discriminative Classification 25

5.1 Support Vector Machines . . . 25
5.2 Logistic Regression . . . 27
5.3 Extending Binary Classification to Multi-class Problems . . . 28
5.4 Normalization of Document Vectors for Classification . . . 29

II Implementation **31**

6 Data Preparation **33**
6.1 Training and Development Corpus . . . 33
6.2 Evaluation Set . . . 34
6.3 Phoneme Transcription . . . 34

7 Baseline System **37**
7.1 Baseline Language Models . . . 37
7.2 Gaussian Backend . . . 37

8 Implementation of iVector System **39**
8.1 Classifier . . . 39
8.2 Document Vector . . . 40
8.3 Standard iVector system . . . 41
8.4 Reset-Trained iVector System . . . 44
8.5 iVector Dimension . . . 46
8.6 System for Shorter Duration Utterances . . . 47

III Results **49**

9 Identification Results **51**
9.1 Baseline system . . . 51
9.2 iVector system . . . 51

10 Detection Performance **55**
10.1 Baseline System on 30 Second Utterances . . . 55
10.2 iVector system 30 second performance . . . 55
10.3 Ten and Three Second Performance . . . 56
10.4 Comparison to Other Systems . . . 56

11 Summary **59**

12 Conclusion **61**

Bibliography **63**

A Properties for the Newton Raphson Updates **67**
A.1 Proof of Symmetry 67
A.2 Conditions for Positive Definity and for Nonsingularity 67
A.3 Column Span of the Total Variability Matrix 68

B Overview of the Implemented Systems **71**
B.1 Scripts 71
B.2 Usage of iVector Program 72

— 1 —
Introduction

Human beings will usually recognize the spoken language from lexical knowledge or familiarization with more subtle language cues [1, p. 785]. Learning to recognize a language requires a prolonged exposure to the language, making it unlikely for any human to familiarize themselves with all of the around 7000 spoken languages in the world [1, p. 798]. As more multimedia content becomes available on the internet and the processing power of machines increases, computers are given the potential to exceed at language recognition to a level beyond what humans are capable of. The technology can in turn be applied e.g. to adding metadata to audio or video databases or as a preprocessing step for multilingual spoken user interfaces.

To realize such a performance, machines will need intelligent learning methods to structure the vast amounts of information available from spoken data. Since some languages are spoken by very few people it might not be feasible for machines to be able to recognize all languages. Still, further research efforts will be necessary in order to enable the machines to identify the over 300 languages that are spoken by more than 1 million people around the world [1, p. 797].

1.1 Problem Description

The purpose of automatic language recognition is to identify which language is spoken from a speech sample. One of the most popular approaches to language recognition is based on phonotactics, i.e. the assumption that the distribution of sound combinations, phone n-grams, is a distinguishing trait of a language. In these systems, a phone recognizer first creates a phonemic representation of the speech. The number of occurrences of phone n-grams in the recognized phone sequence is stored in a spoken document vector.

In the traditional approach, the document vector is scored against statistical phone n-gram models for each language to determine which language is the most likely. An alternative approach is vector space modeling where the document vector is transformed from a pure count to a representation which emphasizes sound combinations distinctive to a language and de-emphasizes combinations which occur uniformly across languages.

Finally, it is possible to use the document vector directly for classification, using discriminative classifiers to separate the languages. This is the topic of this thesis. One problem with direct classification is that the dimensionality of the document vectors is very large. If the number of phones in the phone recognizer is 50, and we count 3-grams, the dimension is 125 000. By nature, the vectors are sparse, since only a fraction of the

125 000 possible 3-grams will occur in e.g. a 30-second utterance. Training classifiers on full-sized document vectors will be computationally wasteful, in addition to being difficult due to the limited coverage represented by any realistic training set. Techniques to reduce the vector's dimension are thus usually employed. iVector subspace modeling is a recent approach to dimensionality reduction that has shown promising results. The compressed vectors, called iVectors, are found through maximum likelihood estimation, making it significantly different from minimum squared error techniques like latent semantic mapping or principal component analysis. The assignment is to examine the iVector approach, construct an iVector language recognition system, and evaluate its performance against more traditional language recognizers.

1.2 Objectives for the Thesis

Our primary goal for this thesis is to construct an high-performing iVector language recognition system. This will require both an understanding of how iVector subspace modelling works, and experiments e.g. to ensure that the published approaches to best train the system will also apply to our training material. Since building a state-of-the-art language recognition system can be an huge undertaking, we will not focus on well-established techniques that should increase the recognition performance of almost any system. Example of such techniques are the use of lattices [1, p. 817] or training the sytem with more material [1, p. 836].

We can use standardized test-sets to compare the performance of the system to other systems but in order isolate the effect of only the iVector modeling, we will need to construct a baseline system. If the two systems share all but the iVector related components, then any difference in performance can be linked to the usage of iVectors. The construction of a baseline language recognition system is therefore essential to evaluate the fulfillment of our primary goal.

1.3 Structure

The rest of the thesis is structured as follows:

- **Part I - Theory**
  - **Chapter 2 - Language Recognition Systems** introduces the goals and the theory behind common components of a spoken language recognition system
  - **Chapter 3 - Statistical Language Modeling** explains the theory behind the language model used in the baseline system

  - **Chapter 4 - iVector Subspace Modeling** gives the theoretical background for iVectors
  - **Chapter 5 - Discriminative Classification** explains the vector based classifiers needed for an iVector system

- **Part II - Implementation**
  - **Chapter 6 - Data Preparation** outlines what data-set will be used for training and testing the systems, and the tokenization of this data
  - **Chapter 7 - Baseline System** discusses the implementation of a PRLM system to compare the iVector system against
  - **Chapter 8 - iVector System** explains design choices for the iVector system

- **Part III - Results**
  - **Chapter 9 - Identification Results** documents the performance of the system on language identification
  - **Chapter 10 - Detection Performance** compares the systems on the language detection task
  - **Chapter 11 - Summary** discusses our results for iVector based language recognition
  - **Chapter 12 - Conclusion** sums up the work in this thesis

Part I

Theory

— 2 —
Language Recognition Systems

2.1 Language Detection vs. Identification

Identification and detection of languages are two very related tasks in language recognition. In both tasks languages are constricted to be in a set of classes. Each class can be a single language, dialect, or a set of languages, and the goal of our system is to separate these classes. In this thesis we will only look at formulations where each class is a single language and possibly one class that consists of all other languages, which we refer to as an *out of set language*. We call it an *open set* recognition problem When the set includes an out of set language, and *closed set* otherwise. With this formulation in mind, we drop the notion of classes, and just call each class a language.

Given a hypothesized language, a language detection system will either accept or reject the claimed language based on a set of observations. The confidence the language detection system requires for its decisions will vary on application, but we say that we accept the hypothesis if the probability that it is the hypothesized language, l_i, given the systems knowledge of languages, θ, and observations, S, is greater than some threshold, t. The acceptance criteria is then defined as

$$p(l_i|\theta, X) \geq t. \tag{2.1}$$

Any claim that doesn't satisfy the equation will be rejected. There are two sources of error associated with a decision in equation 2.1. First we can choose to accept that language l_i was used in the given utterance, when it actually was another language. This is called a *false accept*. The second error type is when we reject the hypothesized language even though it was true, which is called a *false reject*. The tolerance for false accepts and false reject will vary for different applications. From equation 2.1 we see that the errors should be inversely correlated, so there is no universally best threshold-value for any system. E.g. a recognition system that redirects phone customers to an operator that knows the customers language, could want to minimize the number of false accepts, and rather have the caller type in his language when there is much doubt.

In language identification, our goal is just to find the most probable language from the known languages, that is

$$\underset{i}{\operatorname{argmax}}\, p(l_i|\theta, X). \tag{2.2}$$

This means that unless we are implementing a system for a real application, the distinction between the two language recognition problems are not that important so long as we

find $P(l_i|\theta, X)$. What is more important is that the model of the languages, θ, is suited to distinguish languages.

2.2 Evaluation Metrics

As explained in the previous section, it is not clear how we should evaluate the system's performance without knowing the application it will be used for. A simple metric is the percentage of correctly identified utterances by equation 2.2. This score easily calculated and should be correlated with other metrics. A more thorough understanding of the system's performance is given by its *detection error tradeoff*- or DET-curve. This is a plot of the false accept rate against the false reject rate when using different detection thresholds in equation 2.1. The equal error rate (EER) is the point where the system makes just as many false accepts as false rejects. This is a commonly used metric to reduce the information from the DET-curve down to a single number. We have also included the C_{Det} metric which measures the expected cost of making a detection decision. This was the primary evaluation metric in the 2003 NIST language recognition evaluation, and is for each target language defined to be [2]

$$\begin{aligned} C_{\text{Det}} = &(C_{\text{False reject}} P_{\text{False reject}|\text{Target}} P_{\text{Target}}) + \\ &(C_{\text{False accept}} P_{\text{False accept}|\text{Non-target}} P_{\text{Non-target}}). \end{aligned}$$

By adjusting the cost of making different errors, this metric can be used to evaluate systems for a wide range of application requirements. In the NIST evaluation plan both costs are set to 1, and the priors to 0.5.

2.3 Language Characterizations

There are many characteristics from speech that could be used to discriminate languages. The speech features that we use for language recognition is likely to impact the systems performance. In [1, p. 801] a number of sources of discriminative information for spoken languages are given, including:

- **Spectral characteristics.** Languages are made up of different sounds, so it will be possible to distinguish languages based on the acoustic features present in the speech signal.
- **Phonological information.** In a language, a phoneme is the smallest unit of sound that can change the meaning of a word. Different languages use different collections

of phonemes which makes them distinguishable by e.g. the frequency that a phoneme or a sequence of consecutive phonemes occur in a spoken utterance.

- **Lexical information.** Languages are separable by the vocabulary, or set of words, that they use.

A language recognition system doesn't necessarily need to use just one of these sources. In practice, superior performances are reached by using a combination of the language traits for recognition [1, p. 818]. This is because the different traits may yield complementary information about the spoken language. There are different benefits and drawbacks of using the different information sources. Lexical knowledge is often the most important discriminative information source for humans [1, p. 787], and has been used by language recognition systems to achieve excellent performance [1, p. 803]. Unfortunately such systems require speech recognizers for each language making it computationally expensive and thereby impractical when we need to distinguish many languages. For this thesis, we have only looked at phonotactically based language recognition.

2.4 System Overview

In order to put the rest of the chapters into context, we will here present two typical phonetic language recognition systems. The first system depicted in figure 2.1 is called a Phoneme Recognition followed by Language Modeling (PRLM) [1, p. 817]. The speech signal is first turned into discrete tokens or phonemes by a phoneme recognizer. The whole transcript is split into n-grams. We do this so that the language model only needs knowledge about a phoneme in the context of the $n-1$ previous phonemes. As we will see, this can simplify the rest of the components in the system. The n-grams from an utterance is then evaluated against language models for each language. The scores from the language models are then unified and calibrated before the final recognition decision.

Another design for language recognition systems shown in figure 2.2 is based on an utterance's Vector Space Characterization (VSC). The technique is inspired from the information retrieval community where documents are represented by a document vector [1, p. 826]. For language recognition systems we can construct spoken document vectors from the counts of n-grams in an utterance. If the distribution of n-grams is different for languages, then the documents for languages will lie in different regions of the document space. A classifier can then recognize languages based on the region that a document vector is situated in. The size of the document vectors will grow exponentially with n, often resulting in a sparse document vector of a very high dimension. This often

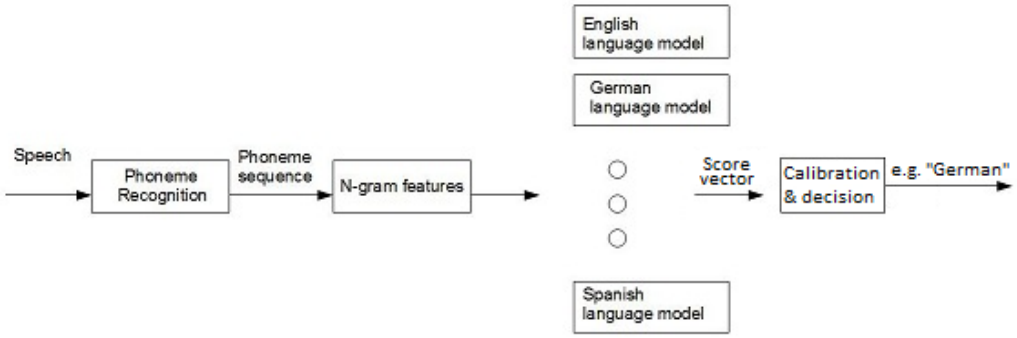

Figure 2.1: Block diagram of a PRLM system, based on [1, Fig. 41.5]

makes it possible to reduce the dimension of the document vectors without the document space losing its discriminative power. An example of such dimensionality reduction is iVectors, which is the topic in section 4

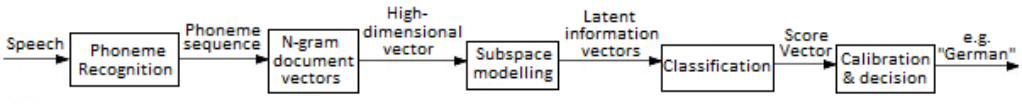

Figure 2.2: Block diagram of a VSC language recognition system. The use of dimensionality reduciton is optional.

It would be possible to construct the two systems using rules to recognize languages. The approach taken by many recent language recognition systems is to build a statistical model from training data. Without speculating on how much time it would take to create a rule based system, a statistical driven system should at least be easier to extend to new languages.

2.5 Phoneme Recognizers

In a PRLM or VSC recognition system, we first transform the speech signal to a phoneme sequence or string. This is done by a phoneme recognizer. With the many-to-one mapping by the phoneme recognizer we hope that there still is enough information about the language identity of the utterance, while the computational requirements for further processing will benefit from the reduced complexity of the problem. Even though each language uses its own set of phonemes, it is not necessary to train the phoneme recognizer to identify phonemes from all languages. This is because the phoneme

recognizer doesn't need to make an correct transcription for an utterance, the system just needs the transcripts to be sufficiently distinct for each language [1, p. 818]. For this reason it is possible to train the phoneme recognizer on a single language, which even may not have to be in the set of languages we are recognizing. Phone recognizers can utilize many of the techniques used in speech recognition, where words often are recognized by their phoneme sequence [3, p. 414]. Here we will only present one such approach.

The speech signal is first split into short frames using a window function to reduce spectrum leakage [3, p. 257]. For each frame we can calculate the Mel-Frequency Cepstral Coefficients (MFCC). The transform takes the spectrum of the windowed signal through a set of filter banks that are spaced to approximate the Mel-frequency scale [3, p. 314]. After taking the logarithm of the filter outputs, we use the discrete cosine transform to get the cepstral coefficients. The whole process approximates the way the human auditory system responds to sound, which means that it should be suited for automatic phoneme recognition as well [3, p. 314]. Patterns in the cepstral coefficients when uttering a given phoneme can then be learned using labeled training data to train Gaussian Mixture Models (this is presented in section 2.7 for recognizing score-vectors).

The size of each frame is limited, in order to make the speech signal in each frame approximately stationary, but this doesn't mean that the neighboring frames have no information about the phoneme uttered in the current frame. Many recent phone recognizers use long temporal context techniques to capture the evolution of the signal outside of the frame [4, p. 8]. One such technique is to concatenate the n cepstral coefficients in the current frame together with the k past and future frames [4, 9], resulting in a $n(2k+1)$-dimensional feature vector. The high number of parameters in the long temporal context feature vector can make it difficult to train the phoneme recognizer without having huge amounts of training material. By assuming that the coefficients in past frames are independent of future frames, we can split the vector into a left and right context. This enables us to first classify each context separately, and then merge the results into a final decision of what phoneme was spoken [4, p. 36]. We can use Gaussian mixture models 2.7 for both classifying steps. The halving of the feature vector then increases the probability of observing similar features in the training set, reducing the need for much training data.

It is also possible for the phoneme recognizer to give multiple hypoteses for the phonemes used in an utterance. Such phoneme recognizers are said to be lattice based. Intuitively a lattice based phoneme recognizer will keep more of the information from the original speech signal, and has been shown to improve the performance of many language recognition systems [1, p. 818].

2.6 Language Models and Classifiers

The responsibility of a language model or classifier is to output a single score per language from the phoneme sequence S. Since our final recognition decision will be based on the probability that an utterance belongs to a language, $p(l_i, |S)$, it seems natural to want the scores to approximate that probability. This is a somewhat strict requirement for systems as described in section 2.4 since the scores will be further processed by the backend. At the very least, a good language model should give scores that are easily transformable to an estimate of this probability.

In order for the system to perform well for languages with limited available training data, it will be beneficial that the system is easily trained. A too complex classifier or language model will not be able to see patterns in the data, it will instead be over-fitted to explain each utterance independently which makes it unfit to score utterances outside of the training set [5, p. 311]. Over-fitting can be reduced by using a larger training set [6, p. 147], but it is clearly an undesirable feature for language models. This is also the reason why we need a separate data set to evaluate the recognition system. The systems recognition performance on the data it was trained for will not necessarily extend to unseen data.

If we introduce the notion that training and test data are generated from a random process, the system's ability to correctly estimate $p(l_i|S)$, and thereby make correct recognition decisions, will be given by two error sources called model *bias* and *variance* [6, 149]. *Bias* is high if the system consistently over- or underestimate $p(l_i|S)$ for some documents regardless of the training set. In this way it represents the systems inability to correctly predict certain documents. Model *variance* is the variance in the probability estimate of an utterance belonging to a language when the model is trained with different data. It represents the systems sensitivity to noise in the training data. Over-fitting is a symptom of a high variance system where the model is only expected to give correct probability estimates for utterances that are very close to a training utterance. A system with limited training data can generally not have both low variance and bias [5, 312], so a concession between the two errors has to be made.

2.7 Backend Calibration of Score Vectors

The backend will calibrate the scores from the classifier or language model into the posterior, $p(l_i|S, \theta)$, so that so that they can be applied to the recognition decisions discussed in section 2.1. This calibration can also be trained using real data. A separate calibration step allows us to ignore application dependent information like what other

languages should the system recognize, and the prior probability of a user speaking each language. The language models for each language can then be trained independently since it only needs to return some number that in some way correlates with the posterior probability for its language. Even if the language model is trained to return estimates of the posterior, there may still be some benefits calibrating the score vectors [1, p. 820] as there might exist patterns in the scores that can be exploited. Furthermore, the backend can be used to fuse scores for an utterance from multiple systems using different phoneme recognizers or language models. The fused system is then expected to perform better than each of the individual systems as long as the errors each system makes are somewhat uncorrelated [1, p. 818].

2.7.1 Gaussian Backend

It is likely that the best calibration method will depend on the nature of the score-vectors given from the language models. A very flexible tool to perform calibrations that impose little demands on the score vector is to use a multivariate Gaussian mixture model (GMM). With this framework, the score vector, $\mathbf{y}$, for a given language is assumed to be produced from a generative statistical model. The likelihood of a r-dimensional Gaussian component k to produce a score vector $\mathbf{y}$ is then

$$p(\mathbf{y}|\mu_k, \mathbf{\Sigma}_k, K = k) = \frac{1}{(2\pi)^{r/2}|\mathbf{\Sigma}_k|^{1/2}} \exp(-\frac{1}{2}(\mathbf{y} - \mu_\mathbf{k})^T \mathbf{\Sigma}_k^{-1}(\mathbf{x} - \mu_\mathbf{k})) \tag{2.3}$$

where $\mu_\mathbf{k}$ is the r-dimensional mean of the score vectors produced by the mixture, $\mathbf{\Sigma}_k$ is the covariance of the mixture and $|\mathbf{\Sigma}_k|$ its determinant [3, p. 94]. From Bayes rule we have that

$$\begin{aligned} p(K = k|\mu_k, \mathbf{\Sigma}_k, \mathbf{y}) &= \frac{p(\mathbf{y}|\mu_k, \Sigma_k, K = k) \cdot p(K = k)}{p(\mathbf{y})} \\ &= \frac{p(\mathbf{y}|\mu_k, \sigma_k, K = k) \cdot p(K = k)}{\sum_{\forall i \in K} p(\mathbf{y}|\mu_i, \Sigma_i, K = i) \cdot p(K = i)} \end{aligned} \tag{2.4}$$

where $p(K = k)$ is the prior probability of the score vector being generated from mixture k. If we know what mixture each score vector in a training set belongs to, then μ_k can be estimated as

$$\mu_k = \mathrm{E}(\mathbf{y}|\mathbf{y} \in k) \tag{2.5}$$

and $\mathbf{\Sigma}_k$ as

$$\mathbf{\Sigma}_k = \mathrm{E}\left((\mathbf{y} - \mu_k)(\mathbf{y} - \mu_k)^T|\mathbf{y} \in k\right) \tag{2.6}$$

where $E(a|b)$ is the conditional expectation [3, p. 94]. The power of the Gaussian mixture model we have now described is that it is enable to model any probability distribution [3,

p. 95], which means that it is applicable to any type of score-vector. Even for a training set, we will generally only know the language of an utterance, but not the mixture the score vector belongs to. This means that equation 2.5 and 2.6 cannot directly be solved. We can still estimate the parameters using the iterative EM-algorithm [6, p. 439]. With one mixture per language, equation 2.5 and 2.6 can be solved by noting that $\mathbf{y} \in k$ is equivalent to that the score vector stems from the given language, l_i, and $p(K = k)$ is the prior for the language. We can then use $p(L = l_i|\mu_i, \Sigma_i, \mathbf{y})$ given in equation 2.4 to label unknown utterances.

— 3 —

Statistical Language Modeling

We will here look at how the baseline system utilizes the phoneme sequences from the phone recognizer to construct language models, which then are used to differentiate languages with other phoneme sequences. This is the only module that the baseline system exclusively uses. Intuitively the best performing system will be determined by the total information loss from its modules, so the performance difference between the two systems will be determined by the information loss from assumptions made in this module versus the iVector specific modules. In section 3.1 we will present how to train a model and use it to decode an utterance. In section 3.2 the concept of smoothing the language models will be presented to make the language model better fit unseen utterances.

3.1 Model training

Using Bayes rule, the probability that the utterance stems from language l_i, will be

$$p(l_i|S) = \frac{p(S|l_i) \cdot p(l_i)}{p(S)}. \tag{3.1}$$

Since $p(S)$ is independent of the spoken language, it only serves as a constant to enforce that the probability of the utterance belonging to any language is 1. The prior distribution of languages, $p(l_i)$, will depend on the application and can be set by the Gaussian back-end discussed in section 2.7. A suited score with no loss of information will be $p(S|l_i)$. An estimate of this score could be found by letting $p(S|l_i, \theta)$ equal the frequency the phoneme sequence appears in the training data for that language. Unfortunately the number of possible phoneme sequences grows exponentially with the sequence length, N. With an almost infinite number of probabilities to measure, it will be impossible to train.

The probability of the phoneme sequence can also be written as chain of random events

$$p(S|l_i) = \prod_{j=1}^{N} p(s_i|S_1^{j-1}, l_i) \tag{3.2}$$

where $S_a^b = s_{\max(a,1)}, s_{\max(a+1,1)}, ..., s_{\max(b,1)}$. We can limit the number of parameters that need to be estimated by making the n-gram assumption, that the probability for the current phoneme will only depend on the $n-1$ previous outcomes. We call the phoneme

sequence S_{j-n+1}^{j} an n-gram at position j. Equation 3.2 can then be approximated to

$$p(S|l_i) \approx \prod_{j=1}^{N} p(s_j|S_{j-n+1}^{j-1}, l_i). \tag{3.3}$$

Phonemes with this model can be seen as drawn from multinomial distributions, with one distribution per phonemes history, S_{j-n+1}^{j-1}. The probability of events occurring in the multinomial models can then be estimated from training data as

$$p(s_j|S_{j-n+1}^{j}, l_i, \theta) = \begin{cases} \dfrac{C_i(S_{j-n+1}^{j})}{C_i(S_{j-n+1}^{j-1})} & \text{if } n \geq 2 \\ \dfrac{C_i(s_j)}{C_i(s)} & \text{if } n = 1 \end{cases} \tag{3.4}$$

where $C_i(S_a^b)$ is the number of times the phoneme sequence S_a^b occurs- and $C_i(s)$ the total number of phonemes in the training data for language l_i. Using these estimates, an unknown utterance can be scored for each language by

$$p(S|l_i, \theta) = \prod_{j=1}^{N} p(s_j|S_{j-n+1}^{j-1}, l_i, \theta). \tag{3.5}$$

Since the first n phonemes in equation 3.5 has less than $n-1$ preceding phonemes, the transition probabilities given less than n previous states has to be estimated as well.

With $C_i(S_{j-n+1}^{j-1})$ draws from a multinomial distribution the variance in our estimate for the conditional likelihood will be

$$\text{VAR}(p(s_j|S_{j-n+1}^{j}, l_i, \theta)) = \frac{p(s_j|S_{j-n+1}^{j-1}, l_i)(1 - p(s_j|S_{j-n+1}^{j-1}, l_i))}{C_i(S_{j-n+1}^{j-1})} \tag{3.6}$$

By definition $C_i(S_{j-n+1}) \leq C_i(S_{j-(n-1)+1})$ so using a smaller value for n will produce a model with less variance. But this will also make the model more biased, as it will only capture short term dependencies between phonemes. Returning to the discussion in section 2.6, n has to be set as a trade-off between model bias and variance. Clearly a too high value for n will make the model unfit to measure the probability of unseen sequences.

3.2 Model Smoothing

A problem with equation 3.5 is that any trigram not observed in the training set will make the probability of observing the whole sequence zero. Furthermore, the granularity of

equation 3.4 is no more than $C_i(S_{j-n+1}^{j-1})$. This means that for rare events, the relative error between the true and estimated probabilities is unbounded. By estimating the probability of rare events with lower order n-grams, the granularity will increase. With only a few observation of an n-gram, it is unlikely that we will find useful dependencies between phonemes spaced far apart, making a high order Markov assumption unnecessary complicated. The method described is known as backoff smoothing [3, p. 559]. The smoothed probability, $\hat{p}$ of a n-gram will be given by

$$\hat{p}(s_j|S_{j-n+1}^{j-1}, l_i, \theta) = \begin{cases} p(s_i|S_{j-n+1}^{j-1}, l_i, \theta) & \text{if } C_i(S_{j-n+1}^{j}) > k_i \\ \alpha_i(S_{j-n+1}^{j-1})\hat{p}(s_j|S_{j-n+2}^{j-1}, l_i, \theta) & \text{if } C_i(S_{j-n+1}^{j}) \leq k_i \end{cases} \tag{3.7}$$

where k_i is some possibly language dependent constant, and $\alpha_i(S_{j-n+1}^{j-1})$ is a constant that makes the sum of probabilities for any n-gram with a given history equal 1. We find α_i by

$$\begin{aligned} \sum_{s_j:C_i(S_{j-n+1}^{j})\leq k} p(s_j|S_{j-n+1}^{j-1}, l_i, \theta) &= \sum_{s_j:C_i(S_{j-n+1}^{j})\leq k} \alpha_i(S_{j-n+1}^{j-1})\hat{p}(s_j|S_{j-n+2}^{j-1}, l_i, \theta) \\ \alpha_i(S_{j-n+1}^{j-1}) &= \frac{\sum_{s_j:C_i(S_{j-n+1}^{j})\leq k} p(s_i|S_{j-n+1}^{j-1}, l_i, \theta)}{\sum_{s_j:C_i(S_{j-n+1}^{j})\leq k} \hat{p}(s_j|S_{j-n+2}^{j-1}, l_i, \theta)} \\ &= \frac{1 - \sum_{s_j:C_i(S_{j-n+1}^{j})>k} p(s_i|S_{j-n+1}^{j-1}, l_i, \theta)}{1 - \sum_{s_j:C_i(S_{j-n+1}^{j})>k} p(s_j|S_{j-n+2}^{j-1}, l_i, \theta)} \end{aligned} \tag{3.8}$$

Since both the granularity and variance should be good for unigrams (1-grams), the values for α_i can be found iteratively for larger values of n by assuming that the unigram frequency requires no smoothing.

— 4 —

iVector Subspace Modeling

iVectors are one example of subspace modeling approaches that can be used to reduce the dimensionality of the data before training and applying classifiers to recognize the language used in an utterance. The dimensionality reduction should make training of the classifiers less computational expensive, which could enable us to train the system with more data. The goal of the reduction is to separate trends in the data that are common to all languages from the information that is unique between utterances. If this separation is properly performed, we will still retain the discriminative information of the utterance in the iVector.

4.1 Background

The idea for iVectors first came from the joint factor analysis (JFA) model used in speaker verification [7]. In JFA continuous speech features are generated from a multivariate Gaussian model

$$\mathbf{M} = \mathbf{m} + \mathbf{V}\mathbf{y} + \mathbf{U}\mathbf{x} + \mathbf{D}\mathbf{z} \tag{4.1}$$

where $\mathbf{y}$, $\mathbf{x}$ and $\mathbf{z}$ are low dimensional normal distributed vectors with zero mean and unit diagonal covariance [7], while $\mathbf{m}$ is the mean distribution vector. By careful training the column span of $\mathbf{U}$ should model the possible effects of channel variability, while $\mathbf{V}$ and $\mathbf{D}$ should model variations in speakers. By having $\mathbf{m}$, $\mathbf{V}$ and $\mathbf{D}$ fixed for all utterances, the speaker dependent information in the utterance would be isolated in the low-dimensional vectors $\mathbf{y}$ and $\mathbf{z}$ which then can be used to recognize speakers. In [7] it was proposed to use a total variability matrix to model the distribution of $\mathbf{M}$ by

$$\mathbf{M} = \mathbf{m} + \mathbf{T}\mathbf{w}. \tag{4.2}$$

The low-dimensional vector $\mathbf{w}$, called an iVector, would now be affected by channel characteristics. Recognition of the speaker could still be performed by first filtering out channel-dependent information in $\mathbf{w}$. This framework was then adapted in [8] to model discrete features by assuming a multinomial distribution where the probability of feature c in utterance n would be

$$\phi_{nc} = \frac{\exp(m_c + \mathbf{t}_c \cdot \mathbf{w}_n)}{\sum_{i=1}^{C} \exp(m_i + \mathbf{t}_i \cdot \mathbf{w}_n)} \tag{4.3}$$

where C is the total number of discrete features, $\cdot$ denotes the inner product, $\mathbf{t}_i$ is the i-th row of the total variability matrix $\mathbf{T}$, and m_i the i-th element of the $\mathbf{m}$-vector. This model was then utilized for language recognition in [9].

4.2 Interpretation of iVector Model

From equation 4.3 we can see the iVector as a set of parameters that govern the probability distribution for features in any utterance. The columns of $\mathbf{T}$ should then span the subspace of likely probability distributions for features in order for the model to fit the actual data [8]. Discrete features like phoneme n-grams should not require us to filter out channel noise, as channel variability should be handled the phoneme recognizer. In this respect, the total variability model should be more fit for discrete features. The mean vector $\mathbf{m}$ is used to move origo in the iVector parameter space. This vector should make iVectors and $\mathbf{T}$ invariant to the mean distribution of document vectors, so that degrees of freedom are only spent on modeling variations in utterances.

Using the assumption of a multinomial model, the log-likelihood of an utterance will be

$$\log(p(\gamma_n|\phi_n)) = \sum_{c=1}^{C} \gamma_{nc} \log(\phi_{nc}) \tag{4.4}$$

where γ_{nc} is the number of times feature c was observed in utterance n. If features are n-grams, then the likelihood given from this model will be quite similar to the model presented in section 3 except that only one multinomial distribution is used per utterance, not one per possible n-gram history. This is a slight inaccuracy in the model when using n-grams, since per definition, only a fraction of the n-grams can follow the previous n-gram. But like JFA, the recognition decision will be based solely on the latent vector, or iVector in our case. We will perhaps model some redundant information since the model requires us to estimate probabilities for n-grams that cannot occur from any context in the utterance. The iVector should make these probabilities low so that there is more probability mass for features that do occur. In any case the iVectors will measure the n-gram probability distribution in the utterance, but using a more general framework than strictly needed [9].

$$\log(p(\gamma|\phi)) = \sum_{n=1}^{N} \log(p(\gamma_n|\phi_n)). \tag{4.5}$$

4.3 Model Training

We can find the parameters for our model by likelihood maximization of equation 4.5. In [9], $\mathbf{m}$ was given the value

$$m_c = \log\left(\frac{1}{N}\sum_{n=1}^{N} \gamma_{nc}\right). \tag{4.6}$$

Now ϕ_{nc} will equal the frequency of feature c when the iVector is all zero. There is no closed form solution to finding values for $\mathbf{T}$ that maximize the likelihood in equation 4.5, but in [9] the problem was given to be concave so that a gradient ascent method could be used to find the absolute maxima. The gradient of equation 4.5 with respect to row c of $\mathbf{T}$ is

$$\mathbf{g}_c = \sum_{i=1}^{N} \left(\gamma_{ic} - \phi_{ic} \sum_{j=1}^{C} \gamma_{ij} \right) \mathbf{w}_i \tag{4.7}$$

This gradient will be all zero at any maxima. We can find this point by using the iterative Newton Raphson method, there new estimates for row c of $\mathbf{T}$ will be

$$\mathbf{t}_c(\text{new}) = \mathbf{t}_c(\text{old}) + \mathbf{H}_c^{-1}(\text{old})\mathbf{g}_c(\text{old}) \tag{4.8}$$

where $\mathbf{t}_c(\text{new})$ denotes the new estiamte for $\mathbf{t}_c$, while $f(\text{old})$ means that the old values for the row should be used. $\mathbf{H}_c$ is the Hessian matrix of equation 4.5 with respect to the cth row of $\mathbf{T}$. In [8] it was proposed to use an approximation of the Hessian to simplify calculations. The Hessian was approximated as

$$\mathbf{H}_c = \sum_{i=1}^{N} \max \left(\gamma_{ic}, \phi_{ic} \sum_{j=1}^{C} \gamma_{ij} \right) \mathbf{w}_i \mathbf{w}_i^T. \tag{4.9}$$

A problem with this method is that the equations depend on the iVector, $\mathbf{w}_n$. This means that we cannot find the maximum in equation 4.5 without both finding values for $\mathbf{T}$ and iVectors. As with $\mathbf{T}$, there is no closed form solution to finding iVectors maximizing the log-likelihood. In [8] the same approach of using Newton Raphson updates were used to find values for iVectors. The gradient for $\mathbf{w}_n$ was given to be

$$\mathbf{g}_n = \sum_{i=1}^{C} \left(\gamma_{ni} - \phi_{ni} \sum_{j=1}^{C} \gamma_{nj} \right) \mathbf{t}_i, \tag{4.10}$$

the approximate for the Hessian

$$\mathbf{H}_n = \sum_{i=1}^{C} \max \left(\gamma_{ni}, \phi_{ni} \sum_{j=1}^{C} \gamma_{nj} \right) \mathbf{t}_i \mathbf{t}_i^T \tag{4.11}$$

making the Newton Raphson update step equal

$$\mathbf{w}_n(\text{new}) = \mathbf{w}_n(\text{old}) + \mathbf{H}_n^{-1}(\text{old})\mathbf{g}_n(\text{old}). \tag{4.12}$$

To find values for $\mathbf{T}$ maximizing equation 4.5, we can do iterations of updating $\mathbf{T}$ and iVectors from a training set. In order to avoid over-fitting the model to the training

data, we can check that an update of $\mathbf{T}$ enables us to increase the likelihood of another set of documents using equation 4.5. We will then have to find iVectors for this set as well, but under no circumstance use these documents to update $\mathbf{T}$. In [9], $\mathbf{T}$ was initialized with small random numbers. This indicates that the updates should converge to the local maxima from most values.

4.4 Extraction of iVectors from the Model

In the previous section we found values for the model parameters $\mathbf{T}$ and $\mathbf{m}$. As a by-product, the training method also found iVectors for the training data. During extraction, when we use the iVector model to find subspace representations of documents, the same process of iteratively finding iVectors using equation 4.12 can be used. This vector should then represent the most important traits of an utterance, and can be used for language classification. Since the classifiers shown in section 5 requires training as well, the iVectors for training utterances will also be needed. It would be possible to use the iVectors found during training of $\mathbf{T}$, but those vectors might be more (or less) converged to the likelihood maxima than the vectors produced during extraction, making them unrepresentative for iVectors found when only performing updates of equation 4.12. A minor point that might benefit the system performance is therefore to discard the iVectors found when training $\mathbf{T}$.

In order for the iVector extraction to be deterministic, the iVectors should be initialized with fixed values. It seems natural to initialize the iVectors as an all zero vector, since origo in the iVector space should correspond to the mean feature distribution of all utterances. The same initial values can be used when we find $\mathbf{T}$ as long as the first step is to update the iVectors. This is because all zero iVectors would cause $\mathbf{g}_c$ and $\mathbf{H}_c$ to be zero as well.

4.5 The Iterative Update Process

Here we are going to look more closely at the Newton Raphson update steps for producing iVectors and the total variability matrix. In a real-time implementation we would need to extract iVectors using these updates for live data which makes the operation time-sensitive. At the same time these updates involve linear algebra on a high-dimension space, making the operations computationally expensive. An inefficient implementation of these updates will therefore severely impact the computational requirements for training and live usage of the system. It is also critical for the total performance of the system

that the iVectors convey meaningful information about an utterance. A thorough study of the Newton Raphson update process is therefore warranted.

4.5.1 *Solving the Newton Raphson Systems*

To avoid issues with numeric instability, it is often desirable not to calculate the inverse of a matrix [10, p. 743]. We can rewrite the linear systems in equation 4.12 and 4.8 to

$$\mathbf{H}_n(\text{old})\delta\mathbf{w}_n = \mathbf{g}_n \tag{4.13}$$

and

$$\mathbf{H}_c(\text{old})\delta\mathbf{t}_c = \mathbf{g}_c(\text{old}) \tag{4.14}$$

respectively where δ means the difference between the new and old vectors. It is beneficial to ensure that these equations have one, and just one, solution. More than one solution would indicate that some of the dimensions in the rows of $\mathbf{T}$ or iVector is redundant, making us solve a more complicated problem than strictly needed. The requirements on $\mathbf{T}$ and iVectors to guarantee just one solution, is shown in appendix A.2. As long as our goal is to find global relationships between utterances, and not over-fit iVectors to each utterance (by letting the iVector dimension approach the number of training utterances or unique features), these requirements should be met. One exception is when we update rows of $\mathbf{T}$ that correspond to features not seen in the training set. Since it is unlikely that we gain much information from such features anyways, we can assume that those rows are always all zero. The rows can then be ignored during iVector updates without much, if any, loss in performance.

In appendix A.2 we also show that the Hessian in equation 4.13 and 4.14 are positive definite. This enables us to use simple algorithms like LU decomposition to solve the systems [10, p. 749]. With an iVector dimension of R, LU decomposition will solve the systems in $\mathcal{O}(R^3)$ asymptotic time [10, p. 750]. While there are faster solvers for positive definite systems like the $\mathcal{O}(R^2)$ solver in [11], R should be of a size that probably doesn't necessitate excessive optimization.

In equation 4.11 $\mathbf{H}_n$ is found by calculating the outer product of rows of $\mathbf{T}$ C times, making the asymptotic runtime $\Omega(CR^2)$. Similarly for $\mathbf{H}_c$ the asymptotic runtime of equation 4.9 is $\Omega(NR^2)$. Since R should be significantly less than N and C to ensure that the linear systems only have one solution, calculating the Hessian will be more computationally demanding than solving the resulting linear systems. In appendix A.1 we show that the Hessian is symmetric, which enables us to only calculate the upper (or lower) half of the Hessians. While the asymptotic runtime remains the same, the actual runtime of should be nearly halved.

4.5.2 *Achieving higher performance*

In appendix A.3 we show that the increase in likelihood by updating iVectors using the Newton Raphson method in equation 4.13 will only depend on two factors. The iVector's values before the update and the column span of the total variability matrix. This means that techniques like having $\mathbf{T}$ orthogonal should give no benefit to the likelihood. There might still be performance benefits when the iVectors are classified as e.g. SVMs are not invariant to linear transforms of the document vectors [12]. However, there are some more promising methods that might increase the performance of the system.

In an iterative algorithm it is important to ensure that each iteration brings you closer to the solution of the problem. The Newton Raphson method is oblivious to high-order derivatives, and we only use an approximation to the Hessian, so an increase in likelihood is not guaranteed from updates using equation 4.13 and 4.14. In [13], Kockmann et. al. would halve the update step until the likelihood from equation 4.5 increased when updating either row of $\mathbf{T}$ or iVectors. If a higher likelihood wasn't achieved after some attempts, the old vector would be used. Seemingly the only downside with such a check would be the additional computational requirements in an update.

In section 5.4 we argued that using unscaled features could cause the classifier to label data only based on high-variance features. This problem might apply to the iVector model as well. A good model of high-variance features would likely be crucial to maximize equation 4.5 and many dimensions in the iVector may then be spent on accurately controlling ϕ for those features. It is not clear if we gain much information from this precise fitting, rather than having coarser knowledge about the exact frequency of those features, and having more degrees of freedom in the iVector to model other features. In [9] the iVector system's performance increased when the square root of elements in the document vector, γ_n, was used. By taking the square root, the dynamic range of high variance features will be heavily scaled, and the importance of modeling each feature should be spread more evenly across the whole document vector.

— 5 —
Discriminative Classification

Discriminative classifiers differ from the generative Gaussian mixture model approach given in section 2.7.1 in that it cannot be used to generate synthetic data. This often results in a simpler optimization problem which is easier to train to give good performance [6, p. 204]. The classifiers we present here can be used on either the full sized document vector or one that is compressed using subspace modeling. We will first look at binary classifiers, where each utterance can take one of two classes in section 5.1 and 5.2, and then extend the technique to multiple classes in section 5.3. Finally we will explore techniques that can increase the effectiveness of the classifier in section 5.4.

5.1 Support Vector Machines

The Support Vector Machine (SVM) has been extensively used and represents the state-of-the-art classifier for text-classification problems [5, 319]. During training, the binary SVM will find a hyperplane that separates the two classes (known as $+1$ and -1 class). Any coordinates, $\mathbf{x}$, that lie on this plane will satisfy

$$\mathbf{w}^T\mathbf{x} + b = 0$$

where $\mathbf{w}$ is the normal vector to the plane and b a scalar. Classification is performed by checking what side of the hyperplane a document, $\mathbf{y}$, lies on by

$$\text{sign}(\mathbf{w}^T\mathbf{y} + b) \tag{5.1}$$

where $\text{sign}(a)$ returns $+1$ if a is positive and -1 otherwise [5, 322]. Since the plane can be defined using both $\mathbf{w}$ and $-\mathbf{w}$, we can choose a planar equation during training so that the output from equation 5.1 can be directly interpreted as the most probable identity of document y . A natural measure of our confidence in the labeling of document y would be the geometric distance between a document and the decision boundary. If the distance is high, then the document is far inside the region where typical documents of the region's class lie. On the other hand, a small distance implies that minor changes in the document vector or the separating hyperplane could result in a different classification. The Euclidean distance between a document and the hyperplane will be the absolute value of

$$\frac{\mathbf{w}^T\mathbf{y} + b}{|\mathbf{w}|} \tag{5.2}$$

[5, p. 323]. This equation is quite similar to equation 5.1 except that it is invariant to the length of the vector $\mathbf{w}$.

During training of the SVM we need to find the hyperplane that separates the training data. For now we will assume that the data is linearly separable, so that there exists at least one such plane. If there exist more than one plane that separates the classes, we would prefer to use the separating plane that has the largest geometric distance from all documents in the training set. Because the normal vector of all hyperplanes can be of any length, we can require $1/|\mathbf{w}|$ to be the minimum Eucledian distance betweeen all training documents and the plane defined by $\mathbf{w}$ and b. This can be stated as the constraint [5, p. 324]

$$c_i(\mathbf{w}^T\mathbf{y}_i + b) \geq 1, \forall i \tag{5.3}$$

where $c_i \in \{-1, +1\}$ is the true class of training document y_i. The hyperplane with the largest geometric margin between any training vector can then be found by minimizing $|\mathbf{w}|$ while upholding the constraints in equation 5.3.

Since it generally cannot be expected that all classes can be linearly separated, there will be some data sets where the constraints in equation 5.3 cannot be fulfilled. Even if the constraints can be fulfilled, there might be some sets where a few unrepresentative document vectors force us to settle with a decision boundary with very little margin between the two classes, just so that we can correctly classify those vectors with any margin at all. Since a single document can have a drastic effect on the decision boundary, the constraints in equation 5.3 will make the classifier have a high model variance. In order to give the learning method less variance, it might be better to have a more biased model that isn't able to correctly classify all documents in the training set. Because of this, SVM often solves an unconstrained problem, where some documents may disregard the constraints in equation 5.3. One such problem is that of a l2 regularized SVM, where we find

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \frac{1}{2}|\mathbf{w}|^2 + C\sum_{\forall i} \max(1 - c_i(\mathbf{w}^T x_i + b), 0)^2 \tag{5.4}$$

where $C > 0$ is a penalty or regularization parameter that can be set to adjust for model bias or variance [14]. Here a penalty is only given if a document fails to meet the conditions in equation 5.3, and when C goes to infinity, the regularized SVM will solve the constrained problem.

For some types of data there might not be any plane that comes close to separate the two classes. For such problems, it is possible to use a nonlinear SVM. Such SVMs first map a document to a higher dimension where it is linearly separable by using a kernel function, and then apply a linear SVM [5, p. 331].

5.2 Logistic Regression

Logistic Regression (LR) is another discriminative classifier where the confidence measure takes a probabilistic and not geometric approach. With two classes, the probability of a given class can be written as

$$\begin{aligned} p(C=1|\mathbf{y}) &= \frac{p(\mathbf{y}|C=1)p(C=1)}{p(\mathbf{y}|C=1)p(C=1)+p(\mathbf{y}|C=-1)p(C=-1)} \\ &= \frac{1}{1+\exp(-\ln\frac{p(\mathbf{y}|C=1)p(C=1)}{p(\mathbf{y}|C=-1)p(C=-1)})} \\ &= \frac{1}{1+\exp(-a)} = \sigma(a) \end{aligned} \tag{5.5}$$

where $\sigma(a)$ is called the *logistic sigmoid* function [6, p. 197], and a is

$$a = \ln\frac{p(\mathbf{y}|C=1)p(C=1)}{p(\mathbf{y}|C=-1)p(C=-1)}. \tag{5.6}$$

The advantage of this seemingly inconvenient expression for $p(C=1|\mathbf{y})$ is that a can take any real value, and $\sigma(a)$ will transform a to a value between 0 and 1. In section 5.1 we argued that the distance from a separating hyperplane could be used as a measure of confidence in our classification. With logistic regression, we can use this distance from a given hyperplane as a measure the log probability ratio [6, p. 205]

$$a = \ln\frac{p(\mathbf{y}|C=1)p(C=1)}{p(\mathbf{y}|C=-1)p(C=-1)} = \mathbf{w}^T\mathbf{y}+b. \tag{5.7}$$

We can then find the hyperplane that maximize the probability of a set of documents belonging to their true class

$$\begin{aligned} \underset{\mathbf{w},b}{\operatorname{argmax}} \prod_{\forall i} p(c_i|y_i) &= \underset{\mathbf{w},b}{\operatorname{argmax}} \prod_{\forall i} \sigma(c_i(\mathbf{w}^T\mathbf{y}_i+b)) \\ \underset{\mathbf{w},b}{\operatorname{argmin}} -\sum_{\forall i} \ln p(c_i|y_i) &= \underset{\mathbf{w},b}{\operatorname{argmax}} \sum_{\forall i} \ln\frac{1}{1+\exp(-c_i(\mathbf{w}^T\mathbf{y}_i+b))} \\ &= \underset{\mathbf{w},b}{\operatorname{argmin}} \ln(1+\exp(-c_i(\mathbf{w}^T\mathbf{y}_i+b))). \end{aligned} \tag{5.8}$$

Given that there exists a hyperplane that separates the classes, we can always find a hyperplane where $p(c_i|y_i)$ can be arbitrarely close to 1 for all the training documents by giving the normal vector, $\mathbf{w}$, an infinite length. As with the SVM we need an incentive to separate the classes with a plane that has a large Euclidean distance to documents in each class. A more suited formulation is to penalize hyperplanes that makes a documents true class improbable using equation 5.8, and maximize $1/|\mathbf{w}|$ so that the log probability ratio

given in equation 5.7 isn't oversensitive to small Euclidean changes in the document vector $\mathbf{y}$. A formulation that should give models with less variance could then be

$$\arg\min_{\mathbf{w},b} \frac{1}{2}|\mathbf{w}|^2 + C\sum_{\forall_i} \ln(1+\exp(-c_i(\mathbf{w}^T\mathbf{y}_i + b))) \tag{5.9}$$

[14] where C is a regularization parameter to avoid over-fitting. Other than using a different penalty-function, this optimization problem is then the same as the one from equation 5.4 that was used to train SVMs. After we've found the hyperplane parameters from a training set, we can use $\sigma(\mathbf{w}^T\mathbf{y}_i + b)$ as an estimate of the probability $p(C = 1|\mathbf{y}_i)$.

5.3 Extending Binary Classification to Multi-class Problems

So far the classification methods that have been described can only differentiate two classes. This is a problem since a general language identification problem will have numerous classes. There exists generalizations of the classifiers that we have discussed that support multiple classes, but these are often avoided due to the additional training time required [15]. A more commonly used approach is to train multiple classifiers on different classes, and then fuse the results to scores for each language. The merging of the results can be done by training Gaussian mixtures discussed in section 2.7.1 to recognize the typical scores from equation 5.2 or the probability scores for the positive class when using LR.

With N classes, the *One vs. Rest* (OVR) approach is to train N classifiers. Each of the classes will act as the $+1$ class in one classifier, and the -1 class in all the other classifiers [15]. The pattern in the scores that should emerge from such a setup is that the classifier trained with a document's true class as $+1$ class should give the highest score. This means that without backend calibration of the scores, a simple way to perform language identification would be to label a vector to belong to the $+1$ class of the classifier that gave the highest score. This is known as max-score wins OVR classification.

Another approach to multiclass classification is to train $N(N-1)/2$ classifiers in a *One vs One* (OVO) pattern. One classifier is then trained for all pairs of classes [15]. Each class is then tested against the $n-1$ other classes as either the positive or negative class. Identification without performing score calibration can be performed by either selecting the class that won most of its $n-1$ trials [15] or the language that received the highest sum of scores.

Although there are more classifiers to train in the OVO approach, this setup should still be faster to train since each classifier require only documents from two classes [1, 832]. The performance of the two methods should be comparable [16].

5.4 Normalization of Document Vectors for Classification

In both the optimization problems for the SVM and LR, we would like to find a hyperplane that separates the data with a large Euclidean distance between documents from the two classes. Dimensions where the document vectors have a very high variance can have a large effect on the Euclidean distance between documents and a hyperplane [12]. This doesn't necessarily mean that those dimensions have more information about the class identity than the dimensions with lower variance. To make the Euclidean distance a more suited confidence measure, document vectors are often whitened by giving each dimension zero mean and unit variance.

After whitening another problem might be that the inner product between the document vector and hyperplane vector can take a very large dynamic range [12]. This can be avoided by giving the whitened document vector, $\mathbf{y}$ unit length. The range of the inner product will then be limited by the length of the hyperplane's normal. Simply dividing the document vector by its euclidean norm would be a many-to-one mapping that looses information. In [12] a more elegant lossless transformation was given where the new score vector, $\hat{\mathbf{y}}$, is

$$\hat{\mathbf{y}} = \frac{[\mathbf{y}^T, D]^T}{|[\mathbf{y}^T, D]|} \tag{5.10}$$

where we concatenate some constant D to each document vector. The scaling of the extra dimension in the new document vector will then reflect the length of the original document vector and can be used to separate document vectors of different lengths. The significance of keeping length-information will likely depend on the distribution of document vectors.

Part II

Implementation

— 6 —

Data Preparation

6.1 Training and Development Corpus

To train the systems we used data from the Linguistic Data Consortium CallFriend corpus[1]. This database contains several hours of unscripted telephone conversations for twelve languages. The languages are shown in table 6.1. For the English, Mandarin

Table 6.1: Languages in the CallFriend corpus

American English	Canadian French	Egyptian Arabic	Farsi
German	Hindi	Japanese	Korean
Mandarin Chinese	Spanish	Tamil	Vietnamese

and Spanish languages the database includes speech from two dialects. The dialects are given in table 6.2. For each language or dialect there is about 60 hours of speech.

Table 6.2: Dialects included in the CallFriend Database

	Dialects	
American English	Non-Southern	Southern
Mandarin Chinese	Mainland	Taiwan
Spanish	Non-Caribbean	Caribbean

Approximately 50 minutes of speech from each dialect was used as development data. This data set is used to test the performance of the system during development. By using a separate data set for the final evaluation, the performance of the system will be given from unseen data. If implementation decisions were to be influenced by the data set used for the final performance evaluation, then the resulting performance might be too optimistic. To make sure that the system recognizes languages and not persons, no speaker was present in both the training and development set. We aimed for having few speakers in the development set in order for the systems to be trained on as many speakers as possible.

(1) http://www.ldc.upenn.edu/Catalog/

6.2 Evaluation Set

For the final evaluation of the systems the 2003 NIST Language Recognition Evaluation set was used[2]. By using a standardized test set, we can easily compare the performance of our system with others. The NIST set mostly includes data collected for (but not used in) the CallFriend corpus [2] so, the conditions for the test segments should be similar to the training data. The NIST set includes the same twelve target languages that were included in the training set given in table 6.1. The set also includes out of set utterances in Russian. The Russian segments can be used to test the system's ability to recognize if any of the trained languages were spoken at all. In accordance with the evaluation rules [2], no attempts were made to prepare the system specifically for Russian out of set segments.

The NIST set includes data of 3, 10 and 30 seconds duration of speech. While our main focus has been on the 30 second segments, we will report results for the other segment lengths as well. The segment lengths were enforced by using an automatic speech activity algorithm to split utterances to the correct size. For each language and duration at least 80 segments from 40 speakers were provided.

6.3 Phoneme Transcription

Phoneme transcription for all the systems were performed using the Brno University of Technology (BUT) phoneme recognizer [3]. The recognizers utilizes the long temporal context features as explained in section 2.5. The phoneme recognizer was trained on Hungarian utterances and has previously performed well on language recognition tasks [9, 17]. With this training, the recognizer distinguishes 62 different classes. The high number of classes can make training material sparse when using higher order n-grams. In order to reduce the sparsity any token labeled as noise were ignored, and consecutive tokens of silence were stripped to just one token. Experiments were also performed using a many-to-one token mapping suggested in [17]. With this mapping, information about phoneme duration is ignored resulting in only 32 different tokens. By itself, we expect this loss of information to negatively impact the performance of the system. However, this mapping can severely reduce the training time for some of the systems. This could potentially mean that we could train the systems with more data, or use higher-order n-grams.

After utterances have been transcribed, we split the transcription into parts so that e.g. the length of a development set utterance match the length of the test set utterances.

(2) http://www.itl.nist.gov/iad/mig/tests/lre/2003/
(3) http://speech.fit.vutbr.cz/software/phoneme-recognizer-based-long-temporal-context

Since the system's performance should be dependent on the utterance length, this should make development scores more closely approximate the score we would expect during final evaluations.

— 7 —
Baseline System

7.1 Baseline Language Models

The baseline system is an implementation of the smoothed language model described in chapter 3. For each language, one model is created for up to phoneme trigrams. The k-parameter that determines the model's degree of smoothing was set independently for each language. A range of parameter values were tested for each language, and the parameter that maximized the log-likelihood in equation 3.5 was chosen. Since smoothing adds bias to the model, the likelihood for the training set will decrease for a higher value of k. Because of this, the total likelihood over the development set was used for the tests. Another strategy might have been to use the same value for k for each language. We could then select the value that maximized the identification rate using equation 2.2 with the likelihoods produced from the model. The former approach was used since trigrams not seen in the training set appeared with clearly different frequency for each language. This indicated that each model would require a different degree of smoothing. It did not seem feasible to maximize the identification rate using different k-values for each language as the parameter search-space would be too great.

The identification rate for 30, 10 and 3 second development utterances all seemed to benefit slightly from using the unmapped phoneme transcript and it was therefore used for the final system. The log-likelihoods produced by the language model was somewhat unsuited to use as score-vector for the Gaussian backend. This is because the likelihoods are affected by the number of phonemes in the utterance. Instead we calculated the posterior probability for each class using equal language priors for the score-vector. This can be viewed as a normalization of the score-vector, making it similar to the score-vector produced by logistic regression.

7.2 Gaussian Backend

For processing of the score-vectors we used the Focal Multiclass Toolkit[1]. The toolkit can be applied to a wide range of machine-learning problems, but it includes methods that are specifically designed for the NIST language recognition evaluation. E.g. find detection threshold values that minimizes the expected cost, C_{det}, of a recognition decision. It also implements training and evaluation using GMMs as discussed in section 2.7. An

(1) https://sites.google.com/site/nikobrummer/focalmulticlass

issue with the backend was that it couldn't be trained on score-vectors from the training set. This is because the language models and classifiers are expected to perform very well when tested on its training data. The score-vectors would then be unrepresentative for the score-vectors we would expect from unseen data. A possible solution would be to introduce a separate training set just for the backend training. Instead of reducing the amount of training material for other system components, we opted for training the backend with the development data. Since we no longer have a set for testing the implementation of the backend, we could no longer experiment with- and compare different backends. For this reason we used simple and easily trained models in the backend that should ensure reasonable results.

For each language, a single Gaussian Multivariate Model was trained. We followed the recommendation given in [18, p. 70] to treat dialects as separate languages in the rest of the system, and fuse the scores for these languages in the backend. This makes the single Gaussian Model an obvious source of bias error, since we expect that score-vectors from utterances of different dialects will be located in separate clusters of score-vectors. On the other hand, we could easily end up with an over-fitted model if we were to represent languages with Gaussian Mixtures.

7.2.1 Universal Background Model for Out-of-Set Languages

The Gaussian backend was also responsible for recognizing the out-of-set language in the evaluation set. It would be possible to train a language model to distinguish most other languages from the target languages. This approach was taken in [18, p. 55] with good results, but it would require training material for an ensemble of out-of-set languages. For our systems, we make the assumption that the score vector from an out-of-set language will be contained in the same region in the score-space as utterances from all the target languages. We train an Universal Background Model (UBM) using all the development data from target languages. The UBM is just another Gaussian Mixture that can then synthesize utterances for any language. Due to the risks of over-fitting, a single Gaussian component was used for the out-of-set language as well. Since the single Gaussian component has to model score-vectors from more than one language, we expect this component to have a higher variance than the components for single languages. This means that the model will give a target language higher likelihood than the UBM if an unknown utterance is scored close to the mean of a "single-language" component. Thus the UBM creates the desired effect of requiring some confidence before we assign an utterance to any target language at all. Still, this simplification is expected to degrade the performance of the system.

— 8 —

Implementation of iVector System

In this chapter we will go over the implementation details of the iVector system. The iVector system will use an equivalent Gaussian backend as the baseline system covered in section 7.2. Since we couldn't test the system using the Gaussian backend during development, most implementation decisions in this chapter were decided with the identification rate of classifier on 30 second development iVectors using the max score wins techinque described in section 5.3.

8.1 Classifier

To classify iVectors, we used the LIBLINEAR library [1]. It is optimized for linear classification on large sets of data and features [14], well documented and supports both SVM-based and logistic regression classification. We normalized iVectors according to the techniques in section 5.4. Using a standard iVector implementation, we tested both the OVO and OVR approach to multiclass classification. We found a slight adaption of the OVR approach to perform best. When a dialect was the $+1$ class, we didn't use the other dialect of the same language in the -1 class. This simpler classification task greatly reduced the identification error rate for the two-dialect languages, while the identification rate for the other languages remained more or less the same. This technique did not extend that easily to OVO classification as dialects would then have less tests than the other languages. The number of wins, or sum of scores for a language would then have to be normalized by the number of classes it was tested against. This more complicated treatment of dialects had little effect on the identification performance which was about the same as we got when using standard OVR.

We tested three approaches to set the penalty-parameter for the regularized classifier. For all three approaches we tested a finite set of parameters against development vectors. In the first test we assumed that each classifier would use the same penalty-parameter, and simply chose the parameter that got the highest total identification rate. For the other two approaches the penalty-parameter was set independently for each classifier the OVR-bank consisted of. This was performed by either maximizing the number of correctly labeled utterances minus the incorrectly labeled, or maximizing the sum of soft scores for a document belonging to its true class. Using the same penalty parameter slightly but consistently outperformed the other methods.

(1) http://www.csie.ntu.edu.tw/~cjlin/liblinear/

In our identification tests, the SVM and logistic regression gave comparable results. The similar iVector system from [9] still reported that the system performed better on language detection tasks when it used logistic regression. This is perhaps not so unexpected if we use backend score-calibration since the soft-scores from the logistic regression is a more meaningful interpretation of the class assignment confidence than the distance from decision boundary we get from SVMs. Although we didn't get to verify this argument, we ended up using logistic regression in our final system.

LIBLINEAR also uses the same interface as LIBSVM [2] which allowed us easily to experiment with non-linear classifiers as well. For the non-linear SVM we followed the developers of LIBSVM's guide to SVM classification [19]. Their recommendation for a non-linear classifier did not yield a performance better than the linear classifiers regardless of the iVector dimension. We expect that with more experimentation we could have gotten better results with kernel based classifiers, as there should be parameters that make some non-linear classifiers behave as if they are linear [20]. This means that the performance of the linear classifier is a lower bound for the performance we could achieve with a non-linear classifier, but the time required to search for SVM parameters did however take considerably longer for the non-linear classifiers. Since it seemed like the iVectors were just as separable with a linear decision boundary, we did not investigate using non-linear SVMs further.

8.2 Document Vector

We use the same process as in 6.3 to generate the phoneme transcripts that are used to construct the document vectors. As expected, the system performed better when no mapping of the phoneme labels was used. Although this result was consistent for all the tests we ran, the savings in computational complexity was significant enough for us to use the mapping for some of our experiments. We used only trigrams as features in the document vector as bigram and unigram counts did not seem to add supplementary information to the vector. In fact the identification rate on 30 second development utterances slightly decreased when we included bigrams and unigrams as features. There was a more significant drop in performance when we used the square root of trigram counts as features. E.g. for a typical iVector-implementation, we experienced more than a 20 % relative decrease in the error rate when ordinary features were used. This result contradicts the findings from the similar iVector system given in [9]. A possible reason for this might be that the other system used phoneme lattices which

(2) http://www.csie.ntu.edu.tw/~cjlin/libsvm/

would make the document vectors richer in information. Without lattices, it might just be harmful to spread the significance of trigrams to those with lower counts.

From the gradient of $\mathbf{T}$ and iVectors given in equations 4.7 and 4.10 we see that a maxima isn't affected by the number of features in the utterance as long as the direction of the document vector is the same. We trained the total variability matrix using utterances with a length of about five minutes of speech. The reason for not matching the training utterance length with the test utterance length was twofold. It is difficult to exploit the sparsity of the document vectors, so having fewer training utterances with more features reduced the time required to train the system significantly. We also expect that the direction of the document vector will be more influenced by noise when the utterance is short. If the total variability matrix were trained on short vectors, then it would also have to model more of this noise that cannot be used to recognize languages. In a way, we might view the shorter development and test utterances as projected into the less noisy subspace when we find their iVectors. This frees up dimensions in the iVector to model more important features. Longer training utterances were also used in the iVector system in [9]. We did not perform an extensive search for the optimal training utterance length, but the identification rate was a bit higher when we used 5 minute rather than 4 or 6 minute training utterances.

After training $\mathbf{T}$ with longer utterances, it might have been beneficial to train the classifier with iVectors from shorter utterances. The classifier would then have more training data available, and the training iVectors would come from utterances of more similar lengths to the test data. This turned out to be true for the mapped phonemes document vectors, where the absolute performance increased slightly when the classifier was trained with two minute long utterances.Without the phoneme mapping, there was no change in the performance, but the classifier ended up with a less regularized model. This might indicate that even though there would be more variance in the short duration iVectors, the decision boundaries between classes weren't much affected. Instead the classifier would more frequently mislabel some of the training vectors. Since shorter training utterances would increase the training time and not provide any significant increase in performance, we opted to use the five minute utterances for training of the classifier as well.

8.3 Standard iVector system

For the basic iVector system we implemented the techniques described in section 4.3 and 4.4. The total variability matrix is then trained with the iterative approach of one update of the ivectors from the train set using equation 4.12 and one update of the

matrix using equation 4.8. Pseudocode for the algorithm is given in appendix B.2.1. We used an iVector dimension of 200. This is substantially lower than the 600 dimension iVectors used in [9], but that system had more training documents, and used phoneme lattices to produce more dense document vectors. The log-likelihood for training and development documents during a typical training can be seen in figure 8.1. In the figure,

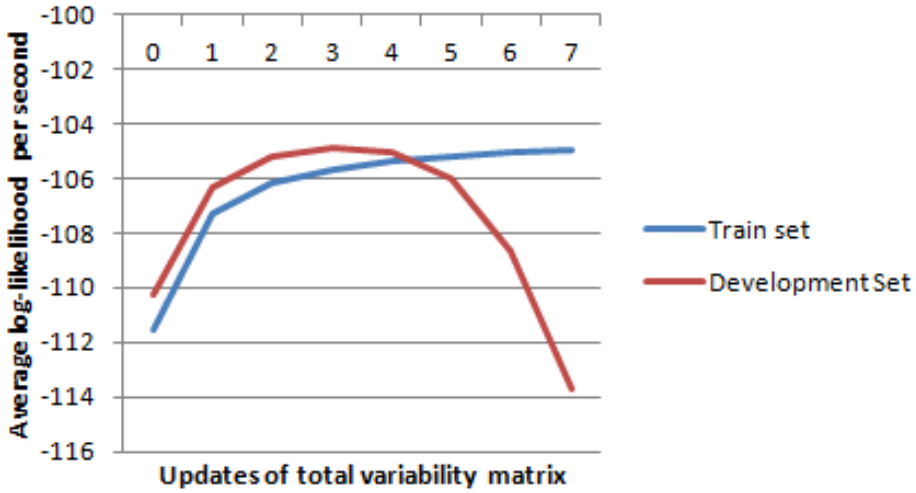

Figure 8.1: Average log-likelihood per second of speech for training and development data during the training algorithm after 0 to 7 iterations. The iVector dimension was set to 200.

the development documents are seen to have a slightly higher log-likelihood than the training documents. This is because we ignore the features in the development document vectors that are not seen in the training set since they would have an likelihood of $-\infty$. The development utterances then seem to have fewer features per second of speech. This effect makes it difficult to directly compare the training likelihoods against development likelihoods, but what we do see is the effect of over-fitting the model to the training data when we use too many iterations of the algorithm. For this reason we stop training $\mathbf{T}$ when a decrease in likelihood for development vectors are observed. In figure 8.1 we would observe an decrease in likelihood during the fourth iteration, so the optimal matrix $\mathbf{T}$ would be the one we found in the third iteration. We also implemented the safeguard against making too large update steps described in section 4.5.2, but it had only a minimal effect on the iVectors.

When extracting iVectors either for training the classifier, or testing the performance of the classifier, we initialize the iVectors to zero. We then do a number of iterations of updating equation 4.12 using the total variability matrix we found during training. Since these are the iVectors that will be used for classification, it seems natural to also take a

look at how the iVectors during extraction react to the trained total variability matrix. In figure 8.2 we show the achieved likelihood for the first 6 iterations of the extraction algorithm for total variability matrices trained in 0 to 7 iterations. Figure 8.2b confirms what we saw in figure 8.1, that too many iterations of updating $\mathbf{T}$ will give the model a worse fit to development utterances. This effect of over-fitting is however not so clear in this figure when the documents are given many iterations to find the likelihood maxima. Still the maximum, regardless of the number of iterations during extraction, is reached when the total variability matrix is trained with around three iterations.

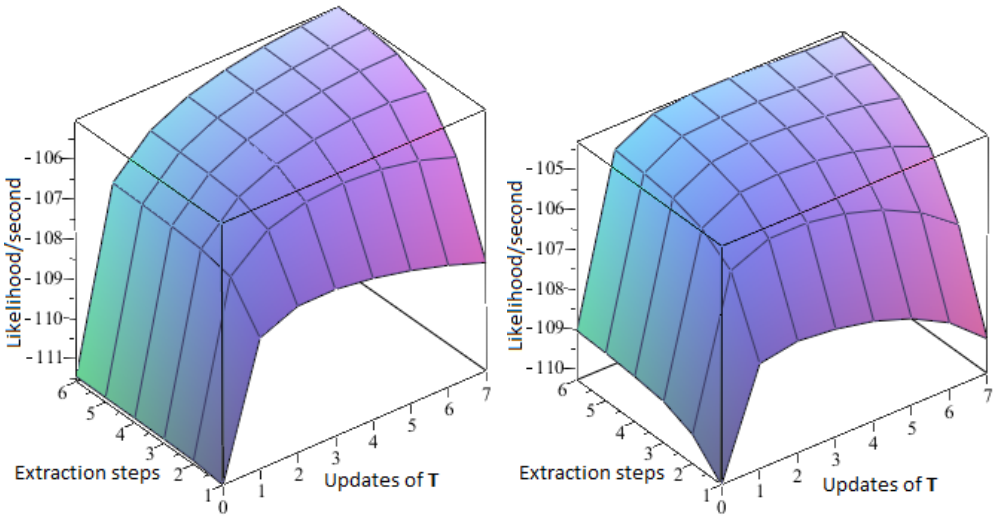

Figure 8.2: Average log-likelihood per second of speech when extracting iVectors with 1 to 6 iterations using a total variability matrix trained in 0 to 7 iterations. (a) likelihood for training documents, (b) development documents. For both tests the iVector dimension was set to 200.

Figure 8.2a showing the likelihood for training iVectors is more troublesome. As expected the figure shows that when $\mathbf{T}$ is trained with more iterations, there will exist iVectors that will give the training data higher likelihood. The problem is that when the iVectors are initialized to zero, it may take many iterations of the extraction algorithm before any benefit of a more fitted $\mathbf{T}$-matrix is shown. With only one iteration of extracting iVectors, the likelihood will actually decrease for training data if $\mathbf{T}$ is trained with more than three iterations. While this decrease in likelihood is only present after $\mathbf{T}$ is over-trained for training utterances, the underlying problem might be a cause of concern as it also may limit the performance of the system when $\mathbf{T}$ is trained with fewer iterations. We suspect the problem to be that as $\mathbf{T}$ gets more fitted to specific iVectors, the Hessian,

or second derivative, to the log-likelihood will change more rapidly in the iVector space. This can make the updates of iVectors converge slower because the Hessian in a Newton Raphson update is approximated to be constant between the old and new vectors.

8.4 Reset-Trained iVector System

One possible solution to the problems with the default system would be to train the total variability matrix using the iVectors we would get during the first few iterations of iVector extraction. Instead of performing iterations of updating iVectors and then update $\mathbf{T}$, each iteration could be to train iVectors from zero and then update $\mathbf{T}$. It seemed natural to try one update of the iVectors per iteration. This keeps the computational requirements during training similar to the standard system, while the total variability matrix is optimized to give documents a high likelihood from the first iteration of the iVector extraction process.

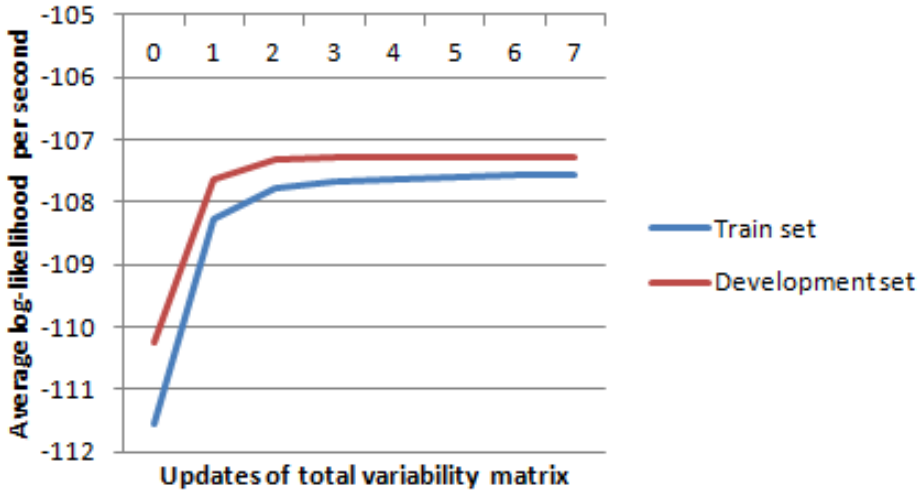

Figure 8.3: Likelihood for training and development utterances during training when iVectors are reset to zero before each iteration. The iVector dimension was 200.

The likelihoods for training and development documents during this training process is shown in figure 8.3. From the figure we see that the likelihood for both training and development utterances ceases to increase after a few iterations and $\mathbf{T}$ never becomes over-fitted to the training data. Still the likelihoods during this training method is smaller than the likelihoods we got using the standard training method, shown in figure 8.1. This is not so unexpected since we restrict the iVectors by resetting them before each iteration.

During iVector extraction, where the iVectors are given more iterations to converge to their maxima, the likelihood for development data even slightly exceeded the maximum likelihood we got from the standard training method.

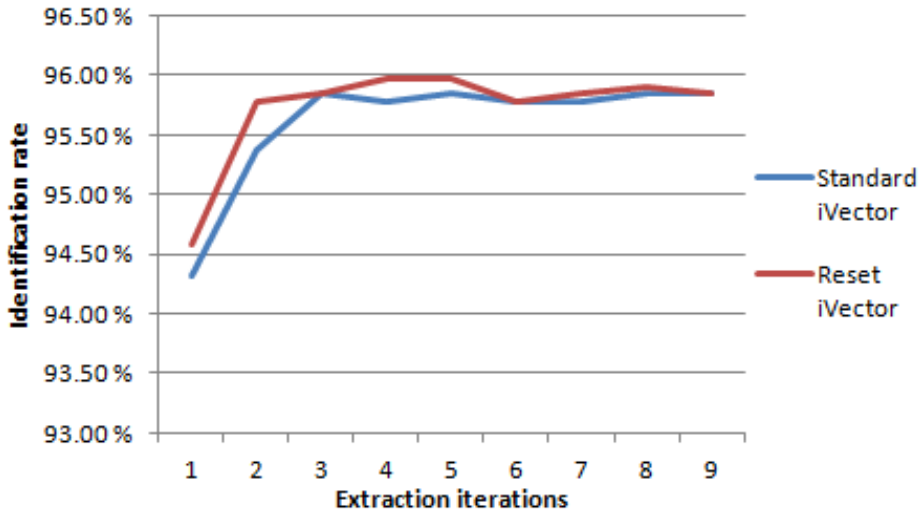

Figure 8.4: Identification rate for 200-dimensional iVectors. The horizontal axis denotes the number of iterations used during extraction to produce the iVectors for training and testing the system

In figure 8.4 we compare the identification rate of the standard and reset-trained systems on 30 second development vectors. As expected the reset-trained system has a higher identification rate when iVectors are extracted with few iterations. It is more surprising that the identification rate for the standard system doesn't exceed the reset trained system at any iteration. We are not sure if this improvement would extend to other data-sets or systems. It might be that the reset-training results in the system having less *model variance*, since $\mathbf{T}$ is trained on iVectors that have only captured coarse details of the utterances. The standard training method on the other hand is able to model minor details in utterances. Although the updates of $\mathbf{T}$ stop before this results in a deterioration of the development likelihood, the modeling of the minor details might offer little discriminative information for unseen data. This slight over-fitting is also supported by the earlier observation that we achieved a higher likelihood for development utterances during iVector extraction, when we used the reset trained total variability matrix.

We also attempted to update the iVectors twice after resetting it in each training iteration of $\mathbf{T}$. Since the iVectors are more closely converged to their maxima, this training method could resemble a middle ground between the standard and reset training. This

did not yield lower likelihoods for development data during extraction.

8.5 iVector Dimension

A very important parameter that hasn't been much discussed is the iVector dimension. Since there will be more degrees of freedom, it seems natural that a higher iVector-dimension will result in lower likelihood for both training and development vectors. Still, this does not mean that we should use iVectors of high dimension. A high likelihood does not necessarily mean that the system will perform better, and using low-dimension iVectors is less computational intensive. In figure 8.5 we show the identification rate for the standard and reset-trained iVector system. We have both trained and tested the classifier using iVectors that where extracted with the same number of Newton Raphson updates. This follows our previous logic of accommodating so that training, development and test iVectors all come from as similar as possible statistical distributions. The number of extraction iterations that gave the best result varied with each system. As we see, the reset-trained system consistently outperformed the regular system regardless of dimension.

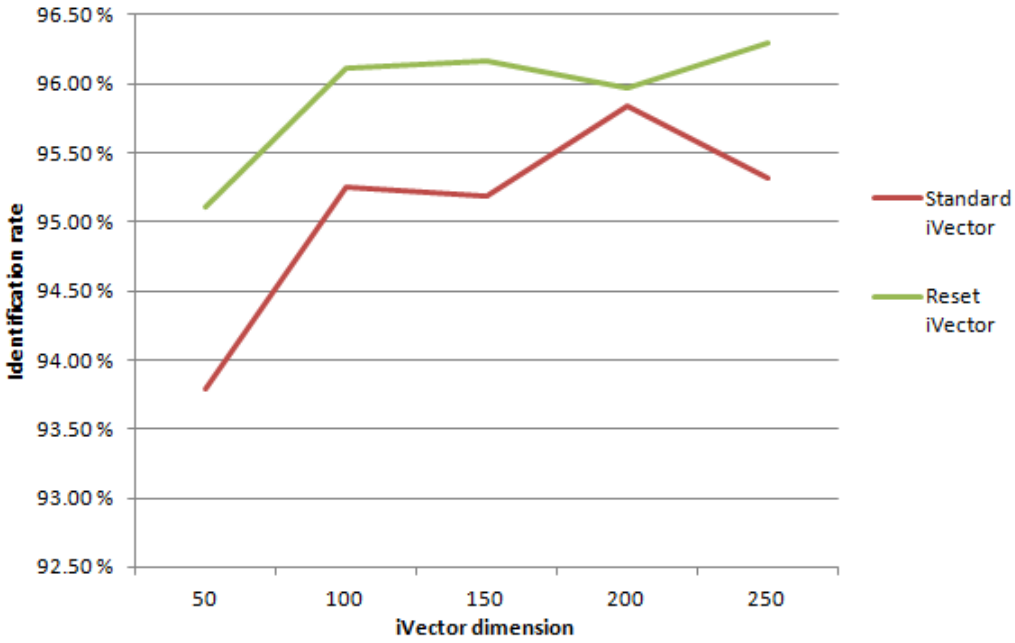

Figure 8.5: Identification rate for standard and reset-trained iVector system with different dimensions. The rates were measured from 30 second development utterances.

It should be noted that the difference between the best and the worst results is only about 35 more correctly identified utterances out of about 1500. We only try a discrete set of penalty-parameters for the classifier, and also plain luck might simply make a few utterances in an iVector-space of some dimension barely cross over the decision boundary to the wrong or correct class. This explains some of the noise in the performance results, and why it is difficult to see clear trends in the performance when increasing or decreasing the performance. The dimension that performed best in the test might not objectively be best suited to separate the languages, and it is no guarantee that it will perform best on the NIST test set. For the final system we used the 150-dimensional reset-trained iVector system. Although the 250 dimensional system performed slightly better on the devtest vectors, the high dimensionality made the system much slower to train and test.

8.6 System for Shorter Duration Utterances

In the past sections we have based our design decisions for the iVector systems performance on 30 second utterances. At least to some degree, many of these decisions should be valid for the 10 and 3 second tests as well. In section 8.2 we argued that the total variability matrix could, and should, be trained using long-duration utterances. For the reset-trained total variability matrix, figure 8.3 shows that over-fitting was not an issue, so it would probably make little difference for the final matrix to check for over-fitting using shorter duration development vectors. All this suggests that the same total variability matrix could be used for utterances of all sizes.

We are however not sure that the same iVector dimension will be optimal for utterances of all lengths. Shorter utterances will have sparser document vectors, and perhaps reduce the need for high-dimensional iVectors. This was not checked, and we used the 150-dimensional reset-trained total variability matrix for the shorter utterances as well. In section 8.2 we found no significant improvement when training the classifier with shorter utterances than what we used when training the total variability matrix. It is possible that there would be clearer performance benefits of using shorter utterances to train the classifiers for 10 and 3 second utterances. The number of utterances will increase if each utterance is shorter, and extracting 3 or even 10 second training utterances for the classifier using the whole training set would simply take too long. A possible solution would be to train the classifier with short utterances from only a subset of the training set, but this was not tested. Since we expect the system to have a worse performance on shorter utterances, the development utterances had the same length as the test utterances. This allows us to optimize the classifiers penalty-parameter for each test.

Part III

Results

— 9 —
Identification Results

For presenting identification results, it seemed beneficial to not use the likelihoods from the Gaussian back-end, and simply select the language that was given the highest likelihood by the language model or classifier. This enables us to compare the results from the NIST 2003 set with the development set. We will here only present the closed set results for the two systems.

9.1 Baseline system

Table 9.1 shows the identification performance of the baseline system on 30 second utterances from both the NIST and development set. For many of the languages there is a big difference in performance for the two sets. This variation seems to be because we used few different speakers in the dev set. For Vietnamese utterances where the error-rate is 0% and 27.7% on the NIST and development set respectively, all the erroneous classifications were from utterances by one speaker. For this speaker only 11 out of 39 utterances were correctly identified. This just illustrates why it was important to train the system on as many speakers as possible, the distribution of phonemes will vary amongst speakers, so it should be beneficial to train the system on a diverse set of speakers. The average identification rate over languages were 93.3% and 91.1% for the NIST set and development set respectively. Also, 93.0% of the 30 second NIST utterances were correctly identified, against 92.1% of the development utterances.

9.2 iVector system

In table 9.2 we show the identification rate of the iVector system. Compared to the baseline system, we got a much higher identification rate on Vietnamese and Tamilian development utterances. German development utterances were however a problem for both systems. In the development set, 96.2% of the utterances were correctly identified, against only 92.8% in the NIST set. This drop in performance for the NIST set, which we didn't see in the baseline system, is probably because the utterances in the NIST-set on average were shorter than the development utterances. Development utterances were split when the phoneme recognizer had transcribed 30 seconds of speech, but the recognizer found on average only 20 seconds of speech in the NIST set. It was not checked whether this was because large parts of the conversation was labeled as noise,

Language	Ar	En	Fa	Fr	Ge	Hi	Ja	Ko	Ma	Sp	Ta	Vi
Arabic	72	0	1	0	0	0	1	0	0	0	0	0
English	2	228	0	4	0	0	0	0	1	1	0	0
Farsi	2	1	72	0	1	1	0	0	0	0	0	0
French	1	0	0	71	0	0	0	0	0	0	0	0
German	0	0	0	1	75	0	2	0	0	0	0	0
Hindi	0	0	0	0	0	72	0	0	0	1	0	0
Japanese	0	0	0	1	4	0	137	1	0	1	0	0
Korean	0	2	2	1	0	4	8	79	2	0	0	0
Mandarin	1	4	1	1	0	3	4	0	75	0	0	0
Spanish	1	1	0	0	0	0	4	0	0	76	0	0
Tamil	1	1	0	0	0	0	0	0	0	1	79	0
Vietnamese	0	3	4	1	0	0	4	0	2	0	1	80
NIST ER %	10.0	5.0	10.0	11.3	6.3	10.0	14.4	1.3	6.3	5.0	1.3	0.0
Dev ER %	1.9	2.4	9.9	7.9	23.8	9.0	3.0	0.0	2.0	5.4	13.9	27.7

Table 9.1: Confusion matrix from identifying 30 second documents from the 30 second NIST 2003 evaluation set with the baseline system. The columns denote the true identity of the document, while rows denote the identified language. ER is the error-rate of the system, which is reported for both the development set and NIST set in the last two rows.

or the automatic speech activity algorithm used to split NIST utterances also included short pauses as conversation. We simply weren't able to see this drop in identification rate for the baseline system, since it had severe trouble with recognizing utterances from some of the speakers in the development set. The average identification rate for each language was 95.5% for the development set and 92.9% for the NIST set

In [21] a SVM based language recognition system using anchor models could correctly identify 90.5% of the utterances in the NIST 2003 set. Both of our systems performed well compared to this result, but the slightly higher performance might however just be the result of using more modern processing of the speech signal.

Language	Ar	En	Fa	Fr	Ge	Hi	Ja	Ko	Ma	Sp	Ta	Vi
Arabic	73	1	1	1	2	2	0	1	0	1	0	0
English	2	223	0	1	0	0	1	0	2	1	0	0
Farsi	1	1	74	0	1	1	0	0	0	0	0	0
French	1	3	0	73	0	0	0	0	0	0	0	0
German	0	0	0	1	74	1	0	0	2	0	0	0
Hindi	1	1	0	0	0	67	2	0	0	0	0	2
Japanese	0	1	1	0	3	0	147	1	1	1	0	0
Korean	1	1	0	1	0	3	2	78	0	0	0	0
Mandarin	0	3	2	1	0	1	4	0	73	0	1	0
Spanish	0	2	0	1	0	2	2	0	0	76	0	1
Tamil	1	3	1	0	0	3	0	0	0	1	79	0
Vietnamese	0	1	1	1	0	0	2	0	2	0	0	77
NIST ER %	8.8	7.1	7.5	8.8	7.5	16.3	8.1	2.5	8.8	5.0	1.3	3.8
Dev ER %	1.0	0.0	4.0	5.0	16.8	5.9	8.0	0.0	1.0	3.0	3.0	5.9

Table 9.2: Confusion matrix from identifying 30 second NIST utterances with the iVector system. The columns denote the true identity of the document, while rows denote the identified language. ER is the error-rate of the system, which is reported for both the development set and NIST set in the last two rows.

— 10 —
Detection Performance

In this chapter we will present the performance for the baseline and iVector system both for open and closed set detection.

10.1 Baseline System on 30 Second Utterances

In figure 10.1 the DET-curve for the baseline system is shown. The system achieved an C_{Det} of 0.0330 for the closed set and 0.0478 for the open set evaluations. From the DET-curve we see that the miss probability cannot be reduced to below about 4% without a severe increase in false alarms. This might be because the Bayesian models are capable of discriminating classes, even though they give poor probability estimates [22]. Usually the score-vector (which was the probability for each class) used in the backend was dominated by one class with a probability close to 1. This made the score-vector contain almost no information for alternative suggestions to the dominating class, so the backend had to make blind guesses if it were to further reduce the miss probability below a certain point. In hindsight, normalizing the score-vector by dividing by the number of phonemes in the utterance, would perhaps avoid this problem. Such normalizing was used in [23]. The EER is seen to be beyond the point where the miss-probability stagnates, and is 4.91% and 4.38% for the closed and open set respectively.

10.2 iVector system 30 second performance

The Det-curves for 30 second utterances recognized by the iVector system is shown in figure 10.2. A noticeable difference from the baseline system is that this system is more adaptable to different requirements for the miss or false alarm probability. Still the C_{Det} is 0.0353 and 0.0494 for the closed and open set respectively which is slightly higher than the baseline system. The EER is however significantly lower, at only 2.76% for the closed set and 3.59% for the open set. The better EER should probably not be contributed to improved recognition capabilities by iVectors, but rather the shortcomings in the score-vector used by the baseline system. The difference between the open and closed set results suggest like we expected that the assumptions made to incorporate out-of-set languages weren't completely valid. E.g. at the threshold used to calculate C_{Det}, 55% of the out-of-set utterances were recognized as target language utterances.

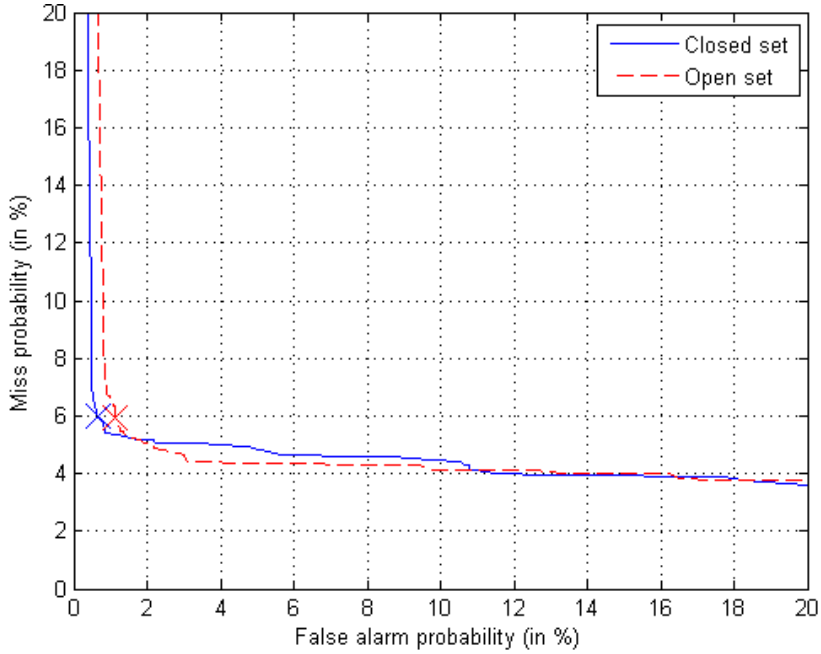

Figure 10.1: Baseline system DET-curves for the 30 second open and closed set NIST utterances. The crosses indicates the miss and false alarm probability used when calculating C_{Det}.

10.3 Ten and Three Second Performance

The DET-curves for the 3 and 10 second tests of the baseline and iVector system are shown in figure 10.3. The iVector system is seen to outperform the baseline system for these conditions. There are only minor differences between the open and closed set results, except for the three second baseline performance. This was caused by the backend recognizing more target language utterances than out-of-set utterances as being out-of-set. Although the assumptions made to accommodate shorter utterances in section 8.6 seems to fit fairly well, we expect that it is possible to improve these results significantly for both systems. This is supported by the comparison of other systems given in the next section, where the EER for our systems are seen to deteriorate faster when the length of the utterances are reduced.

10.4 Comparison to Other Systems

We've listed the performance of the baseline, iVector and 6 other systems in table 10.1. The four systems from MITLL consisted of a trigram PRLM system running in parallel

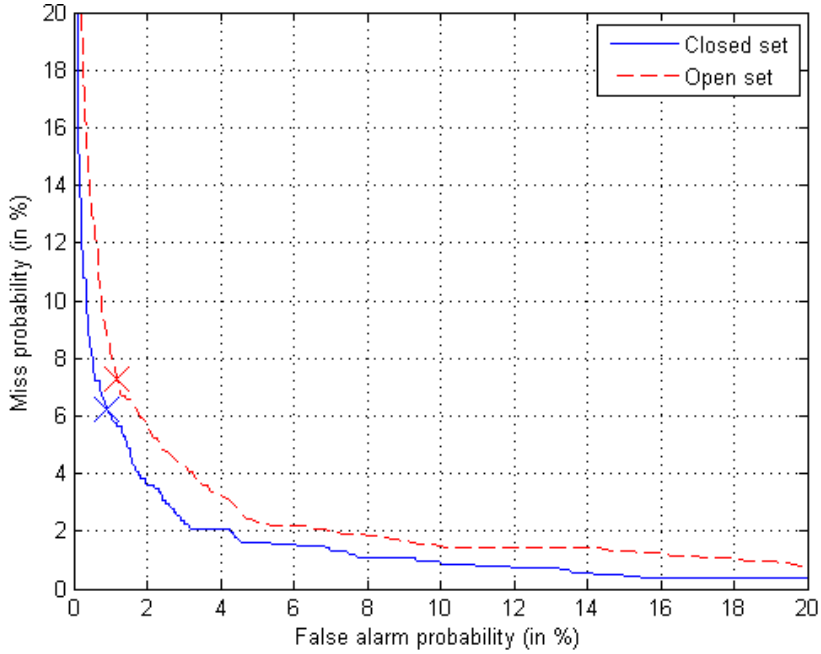

Figure 10.2: iVector system DET-curves for the 30 second open and closed set NIST utterances. The crosses indicates the miss and false alarm probability used when calculating C_{Det}.

with six phoneme recognizer, an acoustic GMM and SVM VSC system and the fusion of the aforementioned systems. The baseline and iVector system performed competitively against these systems for the 30 second test, but they were surpassed by the fusion system for the shorter durations.

System	30s	10s	3s
iVector	2.8	13.3	26.5
Baseline	4.9	18.6	32.0
MITLL-PPRLM	6.6	14.3	25.5
MITLL-GMM	4.8	9.8	19.8
MITLL-SVM	6.1	16.4	28.2
MITLL-FUSE	2.8	7.8	20.3
BUT-SPDAT	2.4	8.1	19.1
BUT-PRLM	1.8	6.6	18.8

Table 10.1: EER in % for an ensemble of language recognition systems for the NIST LRE 2003 closed set evaluations. The MITLL systems are published in [24], while the BUT systems are from [23].

It is difficult to directly compare the iVector approach to other recognition systems

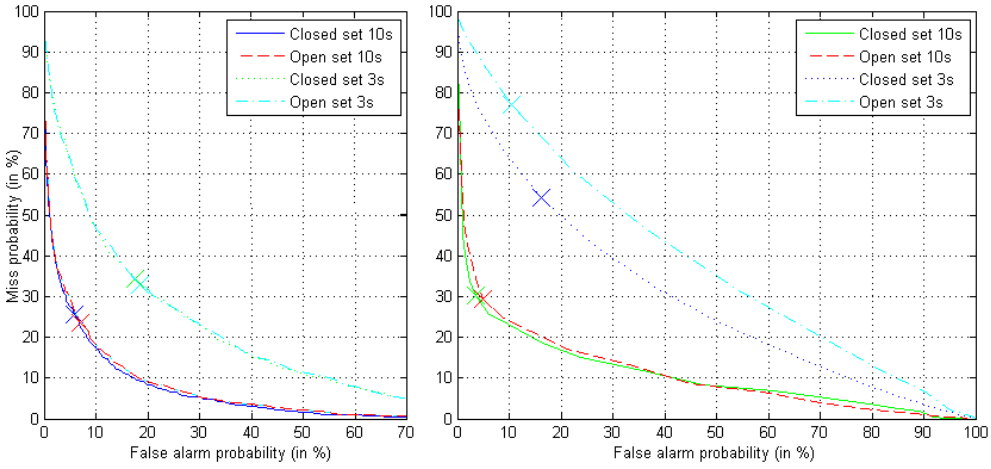

Figure 10.3: DET-curves of the baseline and iVector system for 10 and 3 second NIST LRE03 utterances. (a) is the iVector performance, (b) is the baseline. The crosses indicates the point where C_{Det} was calculated from.

since the performance will heavily depend on the quality of the phoneme transcripts [4, p. 64]. The BUT-SPDAT system used four phoneme recognizers for a trigram PPRLM system, but a superior performance was reached by the BUT-PRLM system. The increased recognition capabilities were partially contributed to better phoneme recognition and use of phone lattices [23]. The phoneme recognizer in the BUT-PRLM system had an phoneme error rate similar to the phoneme recognizer we used [4, p. 58], and lattices were only given to reduce the EER by 0.8% absolute [23]. In light of this, it might have been expected from the discriminative information in our phoneme transcripts to get a higher performance than even the fused MITLL system.

— 11 —
Summary

We got perhaps the most reliable comparison of the iVector system's performance from the identification results, where the baseline's performance was unaffected by the badly normalized score-vectors. Although not necessarily statistically significant, the identification rate for the baseline system was slightly higher for 30 seconds.

Resetting iVectors to zero during training of the total variability matrix seems to offer an interesting alternative to the standard training method to prevent over-training of the model, but it would not be surprising if the learning method is too naive for more complex data. It is possible that tweaking the standard training method to deliberately scale down the change in the iVectors from an update step could result in a trade-off between the two learning methods. I.e. the total variability matrix will at first be trained on iVectors that only portray the rudimentary characteristics of the utterance, but then gradually captures the finer details of the utterance in later iterations until over-training occurs. Even if this has the potential to result in a better subspace it could be time-consuming to experimentally find the best scaling-parameters.

A simpler solution that should in any case increase the performance might be to use more complex data. E.g. the less sparse document vectors we would get from using phoneme lattices. While the standard training method over-trained the total variability matrix after three iterations, the lattice-based iVector system in [9] needed six iterations to be over-fitted. We can't isolate this need for more iterations to just using lattices as that system used other training and development data. Still it seems like more complex data will make the total variability matrix more gradually adapt to the data, which in turn might give finer control over the bias and over-training of the model.

— 12 —

Conclusion

While the final system performed similarly to other well-established techniques for language recognition, it did perhaps not quite reach the performance we would expect given the advantage of using a modern phoneme recognizer. We believe there to be two reasons for this. Some well known techniques to increase a systems performance were left out of scope, but our experiments also suggested that excisting training methods for iVector systems might be better suited for more complicated data.

Bibliography

[1] J. Benesty, M. M. Sondhi, and Y. Huang, *Springer Handbook of Speech Processing*. Springer, 2008.

[2] A. Martin and M. Przybocki, "Nist 2003 language recognition evaluation," in *Eighth European Conference on Speech Communication and Technology*, 2003.

[3] X. Huang, A. Acero, and H.-W. Hon, *Spoken Language Processing; a Guide to Theory, Algorithm, and System Development*. Prentice-Hall, 2001.

[4] P. Schwarz, *Phoneme recognition based on long temporal context*. Doctoral thesis, Brno University of Technology, Faculty of Information Technology, 2008.

[5] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, first ed., 2008.

[6] C. Bishop, *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC, 2006.

[7] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 19, no. 4, pp. 788–798, 2011.

[8] M. Kockmann, L. Burget, O. Glembek, L. Ferrer, and J. Černockỳ, "Prosodic speaker verification using subspace multinomial models with intersession compensation," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[9] M. Soufifar, M. Kockmann, L. Burget, O. Plchot, O. Glembek, and T. Svendsen, "ivector approach to phonotactic language recognition," in *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press/McGraw-Hill Book Company, second ed., 2001.

[11] A. Tewfik and M. Kim, "Fast positive definite linear system solvers," *Signal Processing, IEEE Transactions on*, vol. 42, no. 3, pp. 572–585, 1994.

[12] V. Wan and S. Renals, "Speaker verification using sequence discriminant support vector machines," *Speech and Audio Processing, IEEE Transactions on*, vol. 13, no. 2, pp. 203–210, 2005.

[13] M. Kockmann, L. Burget, O. Glembek, L. Ferrer, and J. Černockỳ, "Prosodic speaker verification using subspace multinomial models with intersession compensation," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[14] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[15] C. Hsu and C. Lin, "A comparison of methods for multiclass support vector machines," *Neural Networks, IEEE Transactions on*, vol. 13, no. 2, pp. 415–425, 2002.

[16] R. Rifkin and A. Klautau, "In defense of one-vs-all classification," *J. Mach. Learn. Res.*, vol. 5, pp. 101–141, Dec. 2004.

[17] P. Torres-Carrasquillo, E. Singer, W. Campbell, T. Gleason, A. McCree, D. Reynolds, F. Richardson, W. Shen, and D. Sturim, "The mitll nist lre 2007 language recognition system," in *Proc. Interspeech*, pp. 719–723, 2008.

[18] P. Matějka, *Phonotactic and Acoustic Language Recognition*. Doctoral thesis, Brno University of Technology, Faculty of Electrical Engineering and Communication, 2008.

[19] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classification." http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf, visited 12th of June 2012.

[20] S. Keerthi and C. Lin, "Asymptotic behaviors of support vector machines with gaussian kernel," *Neural computation*, vol. 15, no. 7, pp. 1667–1689, 2003.

[21] E. Noor and H. Aronowitz, "Efficient language identification using anchor models and support vector machines," in *Speaker and Language Recognition Workshop, 2006. IEEE Odyssey 2006: The*, pp. 1–6, IEEE, 2006.

[22] A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, vol. 752, pp. 41–48, 1998.

[23] P. Matejka, P. Schwarz, L. Burget, and J. Cernocky, "Use of anti-models to further improve state-of-the-art prlm language recognition system," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 1, pp. I–I, IEEE, 2006.

[24] E. Singer, P. Torres-Carrasquillo, T. Gleason, W. Campbell, and D. Reynolds, "Acoustic, phonetic, and discriminative approaches to automatic language identification," in *Eighth European Conference on Speech Communication and Technology*, 2003.

[25] C. H. Edwards and D. E. Penney, *Elementary Linear Algebra*. Prentice-Hall, 1988.

— A —

Properties for the Newton Raphson Updates

This chapter will prove the properties used in section 4.5.

A.1 Proof of Symmetry

From equation 4.11 we have

$$\begin{aligned}
\mathbf{H}_n^T &= \left(\sum_{i=1}^{C} \max(\gamma_{ni}, \phi_{ni} \sum_{j=1}^{C} \gamma_{nj}) \mathbf{t}_i \mathbf{t}_i^T \right)^T \\
&= \sum_{i=1}^{C} \max(\gamma_{ni}, \phi_{ni} \sum_{j=1}^{C} \gamma_{nj}) \left(\mathbf{t}_i \mathbf{t}_i^T \right)^T \\
&= \sum_{i=1}^{C} \max(\gamma_{ni}, \phi_{ni} \sum_{j=1}^{C} \gamma_{nj}) \mathbf{t}_i \mathbf{t}_i^T \\
&= \mathbf{H}_n
\end{aligned}$$

which concludes the proof that $\mathbf{H}_n$ is symmetric. A similar proof using equation 4.9 will show that $\mathbf{H}_c$ is symmetric as well.

A.2 Conditions for Positive Definity and for Nonsingularity

We will here show the conditions for the Hessian of iVectors and rows of $\mathbf{T}$ being positive definite and nonsingular. A nonsingular matrix, $\mathbf{A}$ is a matrix that has one and only one solution to a linear system $\mathbf{Ax} = \mathbf{b}$ [25, p. 54]. Also $\mathbf{A}$ is positive definite if $\mathbf{x}^T \mathbf{Ax} > 0$ for all non-zero vectors $\mathbf{x}$ [25, p. 246]. When we apply this to the approximation to the Hessian from equation 4.11 we get

$$\begin{aligned}\mathbf{x}^T\mathbf{H}_n\mathbf{x} &= \mathbf{x}^T\left(\sum_{i=1}^{C}\max(\gamma_{ni},\phi_{ni}\sum_{j=1}^{C}\gamma_{nj})\mathbf{t}_i\mathbf{t}_i^T\right)\mathbf{x} \\ &= \sum_{i=1}^{C}\max(\gamma_{ni},\phi_{ni}\sum_{j=1}^{C}\gamma_{nj})\mathbf{x}^T\mathbf{t}_i\mathbf{t}_i^T\mathbf{x} \\ &= \sum_{i=1}^{C}\max(\gamma_{ni},\phi_{ni}\sum_{j=1}^{C}\gamma_{nj})(\mathbf{x}^T\mathbf{t}_i)^2 \geq 0\end{aligned} \tag{A.1}$$

The inequality above is given by the fact that ϕ_{nc} and $\gamma_{nc} \geq 0 \forall n, i$ and that $\mathbf{x}^T\mathbf{t}_c$ will be a real scalar. Since the update process explained in section 4.3 only gives iVectors and rows of $\mathbf{T}$ finite values ϕ_{nc} from equation 4.3 will only equal 0 when $m_c = -\infty$. From equation 4.6, this will only happen if the feature, c, is not seen in the set used to calculate m_c. We now let $\tilde{\mathbf{t}}_n$ denote the set of rows from $\mathbf{T}$ that correspond to features seen in the training set or the current vector n, that is

$$\tilde{\mathbf{t}}_n = \{\mathbf{t}_c | m_c \neq -\infty \cup \gamma_{nc} \neq 0\}$$

Equality in equation A.1 can only be reached if $\mathbf{x}$ is orthogonal to all the vectors in $\tilde{\mathbf{t}}_n$. If R is the dimension of rows of $\mathbf{T}$ (and iVectors), then $\mathbf{x}$ can only be orthogonal to the vectors in $\tilde{\mathbf{t}}_n$ if these vectors span a true subset of the R-dimensional space. Each row of $\mathbf{H}_n$ will be a weighted sum of the vectors in $\tilde{\mathbf{t}}_n$, so the row span of $\mathbf{H}_n$ will be a subspace of $\tilde{\mathbf{t}}_n$'s span. If there is an orthogonal vector $\mathbf{x}$, then $\mathbf{H}_n$ will have rank less than R and be singular [25, p. 54]. Since it is given that a positive definite matrix also is nonsingular [10, p. 760], we have that $\mathbf{H}_n$ will be nonsingular and positive definite if and only if the vectors in $\tilde{\mathbf{t}}_n$ span the entire R-dimensional space.

With similar calculations as in equation A.1 we can show when $\mathbf{H}_c$ is nonsingular and positive definite. As with $\tilde{\mathbf{t}}_n$ we let $\tilde{\mathbf{w}}_c$ denote the set of iVectors that contribute (isn't multiplied with zero in equation 4.9) to $\mathbf{H}_c$. Given that we use the same set to train $\mathbf{m}$ and $\mathbf{T}$, γ_{nc} can only be zero if m_c is $-\infty$. So $\tilde{\mathbf{w}}_c$ is either all iVectors from the training set, or the empty set if c is a feature that is not seen in the training set. For features found in the training set, this means that $\mathbf{H}_c$ will be nonsingular and positive definite if and only if all the i-vectors from the training set span all R dimensions.

A.3 Column Span of the Total Variability Matrix

Given two total variability matrices, $\mathbf{T}$ and $\hat{\mathbf{T}}$, with the same column span, there will for any iVector $\mathbf{w}_n$ exist an iVector $\hat{\mathbf{w}}_n$ where

$$\mathbf{T}\mathbf{w}_n = \hat{\mathbf{T}}\hat{\mathbf{w}}_n. \tag{A.2}$$

This is because both sides of the equation will be a linear combinations of the column vectors of $\mathbf{T}$ or $\hat{\mathbf{T}}$ that share the same subspace.

We'll here show that if two iVectors satisfy equation A.2 using some total variability matrices with common column span, then the iVectors we get after a Newton Rapshon update will also satisfy equation A.2. From equation 4.3, this will also imply that ϕ_{nc} and thereby the likelihood when using either of the total variability matrices with the corresponding iVectors will be equal before and after an update. By inserting equation 4.10 and 4.11, the equations for the gradient and Hessian respectively, into equation 4.13, a Newton Raphson update will then be

$$\begin{aligned}
\sum_{c=1}^{C} b_{nc}\mathbf{t}_c\mathbf{t}_c^T(\mathbf{w}_n(\text{new}) - \mathbf{w}_n(\text{old})) &= \sum_{c=1}^{C} a_{nc}\mathbf{t}_c \\
\sum_{c=1}^{C} \left(b_{nc}\mathbf{t}_c\mathbf{t}_c^T(\mathbf{w}_n(\text{new}) - \mathbf{w}_n(\text{old})) - a_{nc}\mathbf{t}_c\right) &= 0
\end{aligned} \tag{A.3}$$

where the scalars

$$a_{nc} = \left(\gamma_{nc} - \phi_{nc}(\text{old})\sum_{i=1}^{C}\gamma_{ni}\right)$$

and

$$b_{nc} = \max\left(\gamma_{nc}, \phi_{nc}(\text{old})\sum_{i=1}^{C}\gamma_{ni}\right).$$

If this equality holds using $\mathbf{t}_c$, $\mathbf{w}_n(\text{new})$ and $\mathbf{w}_n(\text{old})$, then we only need to check that it holds when we instead use $\hat{\mathbf{t}}_c$ and the iVectors $\hat{\mathbf{w}}_n(\text{old})$ and $\hat{\mathbf{w}}_n(\text{new})$ satisfying equation A.2 that we know exists. By using the condition in equation A.2 in equation A.3 we have

$$\begin{aligned}
&\sum_{c=1}^{C} \left(b_{nc}\mathbf{t}_c\mathbf{t}_c^T(\mathbf{w}_n(\text{new}) - \mathbf{w}_n(\text{old})) - a_{nc}\mathbf{t}_c\right) = \\
&\quad \sum_{c=1}^{C} \left(b_{nc}\mathbf{t}_c\hat{\mathbf{t}}_c^T(\hat{\mathbf{w}}_n(\text{new}) - \hat{\mathbf{w}}_n(\text{old})) - a_{nc}\mathbf{t}_c\right) = \\
&\quad \sum_{c=1}^{C} k_{nc}\mathbf{t}_c = 0
\end{aligned} \tag{A.4}$$

where

$$k_{nc} = \left(b_{nc}\hat{\mathbf{t}}_c^T(\hat{\mathbf{w}}_n(\text{new}) - \hat{\mathbf{w}}_n(\text{old})) - a_{nc}\right).$$

Since $\phi_{nc}(\text{old})$ is equal for total variability matrices and iVectors satisfying equation A.2, a_{nc}, b_{nc} and thereby k_{nc} will also be unaffected by the change of matrix and iVectors. We

can conclude the proof if $\mathbf{t}_c$ can be interchanged with $\hat{\mathbf{t}}_c$ in equation A.4 which also can be written as

$$\mathbf{T}^T\mathbf{k}_n = 0, \tag{A.5}$$

where the i-th element in the vector $\mathbf{k}_n$ is k_{ni}. Since the matrices span the same column space, $\mathbf{k}_n$ will either be orthogonal to neither or both the rows of $\mathbf{T}^T$ and $\hat{\mathbf{T}}^T$. Assuming that the system is nonsingular, then equation A.5 will have a solution, and when equation A.2 is satisfied for the old iVectors the nonsingularity will imply that the only possible values for the new iVectors will also satisfy that constraint.

— B —

Overview of the Implemented Systems

We will here give a summary of the scripts and programs developed for the iVector and baseline system. The systems are in no way production ready, and any settings will for the most part be defined in constants. Complete source code can be found at http://www.github.com/asmundto/ivector.

B.1 Scripts

These are the script needed to run the baseline or iVector system.

- *convert.py* will convert the NIST sound-files to the *.raw* format expected for the phoneme recognizer.
- *runphnrec.py* and *NISTrunphnrec.py* runs the phoneme recognizer on speech-files from the CallFriend and NIST set respectively. Requires the BUT phoneme recognizer[1].
- *splitfile.py* splits the phoneme transcripts from the CallFriend set into the desired length. This script will also define the training and development set.
- *baseline.py* trains and then tests the baseline system on development and NIST utterances. The script will report the identification results for both development and NIST data in addition to creating score-vectors for the backend.
- *ivectdocnumvectorizer.py* creates the document vectors required for the iVector system
- *ovarunclassifier.py* trains and performs one-vs-all classification on the iVectors. Usage: *-t <path_to_traindata> -e <path_to_development_or_test_data> -f <0 or 1>*. The first time the script is used on a training set, it will find hyperplane parameters that best separate training data, and the regularization parameters that best explain the development data. Any subsequent calls to the script will use the models already found unless the *-f* option is set to 1. The script will both show max-win identification results and produce score-vectors. Requires LIBLINEAR[2].

(1) http://speech.fit.vutbr.cz/software/phoneme-recognizer-based-long-temporal-context
(2) http://www.csie.ntu.edu.tw/~cjlin/liblinear/

- *lrebackend.m* Uses the score-vectors from development and NIST data to create and test the Gaussian backend and plot DET-curves. Requires the FoCal Multiclass toolkit[3].

B.2 Usage of iVector Program

The iVector program responsible for the subspace modeling is the only component written in C++. It requires the Boost libraries[4] for threading and numeric algebra. While it only has been tested on Windows and Mac platforms it should be cross-compatible. To compile the program using *g++* type:

g++ -Wall -I <path_to_boost_headers> -DNDEBUG -o IVECT Document.cpp FeatureSpace.cpp iVectIO.cpp iVectMath.cpp iVectThread.cpp iVectTrain.cpp log.cpp main.cpp test.cpp Configuration.cpp -L<path_to_boost_compiled_libraries> -lboost_thread -O3

The program has a number of options. While some options can only be changed directly from the source code, the program can be launched with these parameters:

- *-i <dir>* sets the directory to read the file-lists that specify where the location and language of the training, development and test files. These files were created by the document vectorizer.
- *-o <dir>* sets the directory to save the total variability matrix and iVectors.
- *-C <num>* specifies the number of features or dimension of the input document vector.
- *-r <num>* sets the iVector dimension.
- *-s <num>* sets the seed used to initialize the total variability matrix.
- *-L <num>* specifies the column to read features from the document vector files. Usually *1* should correspond to ordinary feature counts, and *2* the square root of counts.
- *-t <num>* the number of threads used by the application.
- *-l <path>* skips training $\mathbf{T}$ and instead loads it from a file.

(3) https://sites.google.com/site/nikobrummer/focal
(4) www.boost.org

Algorithm for Training T
This algorithm computes the iVectors and trained matrix **T** using a set of spoken document vectors from both training and development data with C features. The dimension of the computed iVectors are given by parameter R.

1. Calculate the mean vector, **m**, from equation 4.6
2. Initialize CxR matrix **T** with random numbers, and iVectors as zero
3. Find new iVectors from both training and development set using equation 4.12
4. Check likelihood of development data using equation 4.5. If likelihood has:
   a) increased then continue to step 5
   b) decreased then return the last matrix **T** that did increase the likelihood.
5. Find new matrix **T** using equation 3 with only iVectors from the training set on all rows. Loop to step 3

Table B.1: Standard training algorithm for the total variability matrix.

B.2.1 Algorithms for Training the Total Variability Matrix

The standard training method for **T** is given in table B.1. The reset training would be quite similar, except that in step 3 we would first reset the iVectors from both training and development set to zero before updating them.