Structure Learning in Audio

Nielsen, Andreas Brinch

Publication date:
2009

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Structure Learning in Audio

Andreas Brinch Nielsen

Kongens Lyngby 2008
IMM-PHD-2008-208
By having information about the setting a user is in, a computer is able to make decisions proactively to facilitate tasks for the user. Two approaches are taken in this thesis to achieve more information about an audio environment. One approach is that of classifying audio, and a new approach using pitch dynamics is suggested. The other approach is finding structures between the mixings of multiple sources based on an assumption of statistical independence of the sources.

Three different audio classification tasks have been investigated. Audio classification into three classes, music, noise and speech, using novel features based on pitch dynamics. Within instrument classification two different harmonic models have been compared. Finally voiced/unvoiced segmentation of popular music is done based on MFCC’s and AR coefficients.

The structures in the mixings of multiple sources have been investigated. A fast and computationally simple approach that compares recordings and classifies if they are from the same audio environment have been developed, and shows very high accuracy and the ability to synchronize recordings in the case of recording devices which are not connected. A more general model is proposed based on Independent Component Analysis. It is based on sequential pruning of the parameters in the mixing matrix and a version based on a fixed source distribution as well as a parameterized distribution is found. The parameterized version has the advantage of modeling both sub- and super-Gaussian source distributions allowing a much wider use of the method.

All methods uses a variety of classification models and model selection algorithms which is a common theme of the thesis.
For at en computer kan tage gode beslutninger, er det nødvendigt for den at have så mange informationer om problemet som muligt. To indgange til at få flere informationer om et lydmiljø bliver undersøgt i dette projekt. Den ene indgang er klassifikation af lyden, og i dette projekt bliver en ny metode fundet, der især benytter sig af tone variabiliteten. Den anden indgang er at finde strukterer når flere kilder er mixet sammen i flere observationer. Dette bliver gjort ved at udnytte kildernes statistiske uafhængighed af hinanden.

Tre forskellige problemer inden for lydklassifikation bliver undersøgt. Det ene er klassifikation af lyd i tre klasser, musik, støj og tale. Det andet sammenligner to harmoniske modeller af musikinstrumenter, og deres relevans for klassifikation bliver undersøgt. Og som det tredje bliver en model baseret på MFFC’er og AR koefficienter foreslået til segmentering af populær musik i stykker med sangstemme og stykker uden.


Alle metoder bruger machine learning teknikker indenfor både klassifikation og til sammenligning af modeller, og dette er et gennemgående tema i afhandlingen.
This thesis was prepared at Department of Informatics and Mathematical Modeling, Technical University of Denmark in partial fulfillment of the requirements for acquiring the Ph.D. degree in engineering.

The work was funded by DTU. The project was commenced in October 2005 and was completed in October 2008. Throughout the period, the project was supervised by professor Lars Kai Hansen. The thesis reflects the work done during the Ph.D. project and deals with the extraction of structure and information from audio. This is done through the extensive use and development of various machine learning methods, including Independent Component Analysis, as well as the modeling of audio specific features in the frequency domain.

The thesis consists of the basis theory and a collection of five research papers written during the period 2005–2008 and published elsewhere, and a single technical report not yet submitted.

The thesis is printed by IMM, Technical University of Denmark and available as softcopy from http://www.imm.dtu.dk.

Lyngby, October 2008

Andreas Brinch Nielsen
Papers included in the thesis


A big thank you to my supervisor Lars Kai Hansen who supported me starting the ph.d., and for having great discussions along the way. Also to the Intelligent Signal Processing group for invaluable conversations of technical nature, but certainly also for all the other interesting topics we have discussed over lunch.

For the tedious work of proofreading the thesis I would like to thank my sister Anna, my father, and my friends Bjørn and Rasmus. Thanks to these people, the thesis has become much more pleasant to read.

Last, I would like to thank my wife Cecilie, for supporting me in hard times and especially for taking care of me in the last few weeks before the deadline.
Contents

Summary i

Resumé iii

Preface v

Papers included in the thesis vii

Acknowledgements ix

1 Introduction 1

1.1 Scientific contribution 3

1.2 Structure 4

1.3 Mathematical notation 4

2 Classification 7

2.1 Linear classification 9

2.1.1 Multiple classes 10

2.1.2 Skewed class sizes 12

2.2 Artificial neural network 13

2.2.1 Complexity control 15

2.3 Kernel methods 16

2.3.1 Factor models in kernel space 18

2.3.2 Reduced kernel orthonormalized partial least squares 20

2.4 Bayesian, generative classifier 20

2.4.1 Probability theory and Bayes Theorem 21

2.4.2 Modeling 22

2.5 Probabilistic interpretations of classification models 24

2.6 Summary 25
## CONTENTS

3 Independent Component Analysis ........................................... 27
  3.1 Principal Component Analysis ........................................... 28
  3.2 Independent Component Analysis ....................................... 29
    3.2.1 Maximum likelihood ................................................... 31
    3.2.2 Gaussian mixture ICA ................................................ 33
    3.2.3 Fast ICA ................................................................ 35
  3.3 Summary ........................................................................... 37

4 Model Selection ................................................................. 39
  4.1 Assessing classification performance .................................... 40
    4.1.1 Occam’s razor ............................................................... 41
  4.2 Forward selection/backward elimination ............................... 42
  4.3 Saliency measures .............................................................. 44
  4.4 Bayes Information Criterion ............................................... 46
  4.5 Akaike’s Information Criterion ............................................ 48
  4.6 Automatic Relevance Determination ..................................... 49
  4.7 Summary ........................................................................... 51

5 Audio modeling .................................................................... 53
  5.1 Frequency analysis / spectrogram ....................................... 53
    5.1.1 Fourier analysis ............................................................ 54
    5.1.2 Spectrogram ................................................................ 56
  5.2 Harmonic signals ............................................................... 60
    5.2.1 Pitch estimation ............................................................ 61
  5.3 MFCCs ............................................................................. 65
  5.4 Multi speaker situations ..................................................... 66
  5.5 Summary ........................................................................... 67

6 Audio classification .............................................................. 69
  6.1 Pitch based sound classification ........................................... 69
  6.2 On the relevance of spectral features for instrument classification 74
  6.3 Vocal segment classification in popular music ....................... 79

7 Structure Learning ................................................................ 87
  7.1 Synchronization and comparison of Lifelog audio recordings ....... 87
  7.2 Structure learning by pruning in ICA ..................................... 94
  7.3 Using Gaussian Mixture Models in pruned ICA ....................... 105

8 Conclusion ........................................................................... 123
  8.1 Future work ....................................................................... 124

A Neural network .................................................................... 125

B Optimal brain surgeon ......................................................... 127
Bibliography 129
Chapter 1

Introduction

The number of computers is increasing day by day. Not only personal computers increase in numbers, but more and more appliances and systems are controlled by processors. The amount makes controlling each computer actively hard if not impossible, which results in a ‘movement from human-centered to human-supervised (or even unsupervised) computing’ [83]. This means that computing must be more and more autonomous while the goals will remain to be defined by humans. Autonomous computing means not just acting on input by humans but foreseeing needs and acting on them. By Tennenhouse [83] this is termed proactive computing, and will be at the heart of computing development in the future. This means a change from placing ‘human beings in the loop’ to a situation in which ‘humans are above the loop’.

In order to have widespread proactive computing that seriously influences our lives, extensive modeling of our lives is needed, and this needs extensive data of our lives. Therefore proactive computing is linked to pervasive computing, which is the notion of computers appearing in more and more places, in the open as well as hidden. Most people now a days carries a cell phone. These devices have evolved from a simple phone used for calling other people to small computers, carrying information about contacts, calling history, calendar appointments and with the GPS the location and possible patterns of the user.

One direct motivation for the project are the problems encountered by hearing
aid users. Knowing what kind of situation a user is in gives the chance of optimizing the audio experience for that specific environment, instead of just optimizing for the common case or to react afterwards to input. In most hearing aids in use today the users have to select programs manually, but as this is a problem to many of the elderly it would be beneficiary to do this automatically. This must be based on knowledge about the situation like these need extensive knowledge about the environment and about audio features [16].

Audio classification is at the heart of audio information retrieval, and has been done in connection with many applications. Within music it is used to find the genre of the music [50 8 57] or to identify which instrument is playing [90 54 82]. It is also the basis of speech recognition [72] which is used widely today for example in cell phones, and in office applications. Live broadcasts are subtitled using speech recognition [7]. In [58] a speech recognition system is build that allows searching in CNN News broadcasts without the need of manually tagged information, and this could be used more broadly to allow search in general in audio. Audio classification is also used in surveillance and security related applications [1 73], where emotional outbursts [86 61] might be a sign of a security situation in the making.

In the domain of modeling one cannot avoid the concept of machine learning. Audio physics and room acoustics are much too complicated to model with enough accuracy, and the perception of humans are so complex that it would be impossible to model it using a physical approach. Therefore you need mathematical approaches that can learn structures in data and create models under the supervision of humans. This modeling of data given overall goals is what is entailed in the concept of machine learning. Saying that the physics are too complicated does not mean that you cannot draw inspiration from nature. This is done heavily and has lead to various methods in different areas. An example is the artificial neural network which was inspired by the neurons in the brain. Another example is the feature called the mel-frequency cepstral coefficient, which is motivated in the way the ear processes sounds.

The causal interpretation is a very interesting approach to the modeling of data, and it answers the question of what causes what. This conclusion is much more appealing than simply concluding that an observation occurs together with another. Care must be taken when drawing causal conclusions because sometimes the data simply does not justify them. A common example is the relation between ice cream sales, and drowning accidents, which could be heavily correlated. Of course you cannot conclude that buying an ice cream increases the risk of drowning. The reason for the correlation is explained by them having confounding variables, which could be the weather seasons or simply being at the beach.
1.1 Scientific contribution

Even though causal interpretations can be controversial they are done all the time. It is generally accepted that eating a chili causes a burning sensation in the mouth, and in many other cases it requires skill to explain away the causal relationship. Even in cases where the relations are not apparent you can use the causal methods to give better hypotheses about what is the cause of what even though it might be hard to strictly prove the relation.

A method related to causal interpretation is structural equation models (SEM) \[68\]. They are based on Gaussian distributions and compares different hypotheses based on the correlation structure between the observed and additional modeling variables. The SEM assumes independence through uncorrelatedness of the variables, and as explained in chapter 3 the Gaussian distribution is not adequate to explain such relations. It is possible to identify such relational networks uniquely only if they are sparse enough.

1.1 Scientific contribution

The objective of the current project has been to attain information about audio environments and to develop new methods of processing audio to achieve information. The first approach has been to describe audio using the pitch structure of harmonic sounds. It is shown that extracting the pitch can be used even for non-harmonic sounds to reveal information about the sound and in chapter 6.1 it is used to classify a piece of audio into three classes, music, noise and speech. The classification is based on the pitch dynamics and the pitchness of the sound, which is found to be unique for the three classes. In chapter 6.2 two different models of the envelope of the pitch of musical instruments, and especially how it evolves over different pitches is compared and used to identify which instrument is playing in a small sound bit. In chapter 6.3 popular music is segmented into parts with voice and parts without voice.

The other approach in the project is to extract structure in mixings of multiple sources. The possibility of using simple features to compare recordings to identify if the same sources are present in them is investigated in chapter 7.1. A fingerprinting procedure attains a high classification rate and is computationally very efficient. It could possibly be used on many hours of recordings of multiple observations. In chapter 7.2 a more general approach is developed which can find the same structure in the mixings, but also has the capability of identifying asymmetric mixings. This can potentially be used in relation to causal discovery using the assumption that the sources are independent of each other. The method was extended in chapter 7.3 to a parametric source distribution model, which allows nearly any non-Gaussian distribution. Both methods are
investigated thoroughly under many different settings.

1.2 Structure

The thesis is structured by having the first four chapters describe the models which form the basis of the developed methods. The primary results are then presented in the last two chapters followed by the main conclusions.

Chapter 2 Different classification models are presented, including linear classification, artificial neural networks, kernel methods and the Bayesian classifier.

Chapter 3 This chapter is about Independent Component Analysis and the different approaches of solving the linear mixing problem is presented.

Chapter 4 Model selection algorithms is used to decide the dimensionality and complexity of the models in chapter 2 and 3. The methods presented are forward selection, backward elimination, optimal brain damage/surgeon, BIC and AIC and automatic relevance determination.

Chapter 5 Modeling specific to audio is described in this chapter with emphasis on frequency based approaches and pitch.

Chapter 6 This chapter includes three papers on audio classification.

Chapter 7 This chapter present three papers on learning structure in linear mixings of multiple sources.

Chapter 8 A summary of the main findings and conclusions followed by a discussion.

1.3 Mathematical notation

Mathematical notation is used throughout the thesis, and the general guidelines for understanding it are given here.

Variables will be given as italic letters, $a$, vectors are denoted as lower case bold, $\mathbf{a}$, and matrices in upper case bold characters, $\mathbf{A}$. Entries in a matrix are given in upper case italic characters, the $i,j$'th entry of matrix $\mathbf{A}$ is denoted as $A_{ij}$. The first index, $i$, is the row index and $j$ is the columns index. This means that
A_{ij} would be a matrix with indices ij and not the i, j’th entry in A. Vectors are always column vectors and the transpose of a vector \( \mathbf{a} \) is denoted \( \mathbf{a}^T \).

Many algorithms depend on a sequence of observations and the index into the sequence will be done using the index, \( n \). Superscript will be used for sequence indices in the following way, \( x^n \). Structural indices are given as subscripts, and the \( i \)'th dimension of the \( n \)'th observation is denoted \( x^n_i \), which leads to the following convention, \( \mathbf{x}_{d \times 1} = [x_1, x_2, \ldots, x_d]^T \), and \( \mathbf{X}_{d \times N} = [\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N] \).

\( \mathbf{I} \) is the identity matrix. The inverse of a matrix is denoted \( \mathbf{A}^{-1} \), as mentioned the transpose is \( \mathbf{a}^T \), \( |A| \) means the absolute determinant and \( \mathbf{A}^\dagger \) is the pseudo inverse. \( \mathcal{L} \) is used for the negative log likelihood and \( \mathcal{E} \) is used for other error functions. \( E\{\} \) is the expectation operator. \( \mathbf{\theta} \) is used to denote parameters in general or a collection of parameters. Where possible \( w \) is used for specific parameters, but other letters are used as well, depending on common conventions or to ease readability. \( x \) is used for input data and \( t \) for targets. \( \hat{t} \) is an estimate of \( t \).  \( \mathbb{R} \) is the set of real numbers. \( \mathcal{O} \) is used to describe complexity.
A major part of audio classification is of course the classification algorithms. In general, one does not want the final results to be influenced by the classification framework being used, and therefore it is advantageous to use multiple classifiers, either to integrate the results to increase performance or to verify that the results are not dependent on the particular classification framework being used. A number of approaches have been used for the work in the project, and the focus of this chapter is to present the basic theory of these.

The linear classification framework is the most basic framework, and is probably the most widely used as well. It is limited in classification performance on its own, but since the linear class has so nice analytical properties, it has been extended in various ways and has shown its applicability within many fields. Sometimes a standard preprocessing is used, like adding the squares of the inputs to the input vector, and an extreme example of this is the kernel method, where infinitely many extension are added. Another approach is to tailor the preprocessing to the data leading to feature extraction, like for example in [64].

An early example of a simple artificial neural network was the perceptron [74], but the networks first became popular with the increasing computational capabilities and the invent of the error back-propagation procedure [76]. The brain consists of a large number of neurons which receive a number of inputs and based on them the output of the neuron is either either activated or not. The artificial
neural networks are motivated by this strategy and are capable of modeling any relation between an input and an output \cite{14, 40, 45}. Of course this is a very nice feature but, as always, comes with a drawback in terms of overfitting and complexity.

The method of kernel methods also has the ability of modeling complex relations in data, but has drawbacks in terms of complexity and storage requirements. A particular extension, rKOPLS, amounts to a kind of feature selection in infinite space and is capable of handling the large data sets that can be a problem for other kernel methods.

The Bayesian framework is a theoretically very nice approach to classification (and regression), because it describes the basic probabilistic relations with which any uncertainty in a model can be described. It describes an optimal approach to making decisions and many of the other classification methods can be described in this framework. Unfortunately, strictly applying the Bayesian rules, which leads to the optimal decisions, is in many cases infeasible and approximations and assumptions are used to simplify the problem.

All of the classifiers have been used in at least one of the included papers and will be presented in the following sections.
2.1 Linear classification

In general classification is the procedure of assigning a data point to one of a set of discretely defined classes. This is done by modeling the relationships in a training set, consisting both of inputs and the class labels of those inputs. The model can then be used to classify future inputs for which the class label is unknown. An example which will be used throughout the chapter is that of classifying a piece of audio into one of two classes - music and speech. The data was collected by Nielsen [62] and consists of 28 features based on the pitch of the audio, but for illustration only two features will be used in this chapter. In figure 2.1 a scatter plot of the data is shown. As is evident some areas are inhabited by both classes and the task is to find the regions that divide the area into the two classes.

2.1 Linear classification

Linear classification seeks to find a line separating the two classes. A line is defined as \( \mathbf{w}^T \mathbf{x} = 0 \), where \( \mathbf{w} \) is a vector defining the line and \( \mathbf{x} \) is a point in the plane. By adding a one to the vector \( \mathbf{x} \) an offset is allowed so that the separating line does not necessarily go through the origin, \( \mathbf{x} = [1, x_1, \ldots, x_d]^T \), where \( d \) is the dimension of the data set, \( d = 2 \) in the present example. The labels or targets, \( t \), are defined as \( \pm 1 \), in this example \( t = -1 \) for the music class, and \( t = +1 \) for the speech class. A new point, \( \mathbf{x} \), is classified by projecting it onto \( \mathbf{w} \),

\[
\hat{t} = \mathbf{w}^T \mathbf{x}.
\]

\( \hat{t} \) is the approximation of the true label, \( t \). If \( \hat{t} \) is positive, the point is placed on the positive half plane and the point is classified as the \( +1 \) class, speech. If the projection is negative the point lies in the negative half plane and the point is classified as the \( -1 \) class, music.

We have shown how to classify a point given \( \mathbf{w} \), but we need to specify the values of the parameters in \( \mathbf{w} \), for which we will use a training set. An error function, \( \mathcal{E} \), will be minimized on the training set. The most obvious error function would be the number of misclassifications,

\[
\mathcal{E}^m = \begin{cases} 
1 & \text{sign}(\hat{t}) \neq \text{sign}(t), \\
0 & \text{sign}(\hat{t}) = \text{sign}(t), 
\end{cases}
\]

but the problem with such an error function is that it is not smooth or differentiable so it is very hard to train the parameters of \( \mathbf{w} \). A popular choice of error function is the quadratic error function, \( \mathcal{E} = \sum_n (t^n - \hat{t}^n)^2 \), which by differentiation leads to a closed form solution for \( \mathbf{w} \),

\[
\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}.
\]
where the inputs and targets are concatenated, \( X = [x^1, \ldots, x^N] \) and \( T = [t^1, \ldots, t^N] \), for a training set of size \( N \). \((XX^T)^{-1}X\) is called the pseudo inverse of \( X \) and is sometimes denoted as \( X^\dagger \).

In figure 2.2 the vector, \( w \), and the decision boundary is shown for the two-class problem. The linear separation looks reasonable, but we can see that some points are misclassified because of the overlap in the two classes, but some are misclassified because of the constraint of a linear decision boundary.

### 2.1.1 Multiple classes

The positive/negative setup of the targets only allows for two-class problems, and often more than two classes are encountered. One approach to solve the multiple class classification problem is solving many binary problems and integrating the results. This can give ambiguous results and therefore a direct approach will be presented instead.

The classes are labeled from 1 to \( C \), with \( C \) defining the number of classes, and the \( n \)'th point has the class label, \( c^n \in [1, \ldots, C] \). The target of a data point is now a vector, \( t \), and is specified by a 1-of-\( C \) encoding, where only one entry is 1, the \( c \)'th, and the rest is zero. A \( C \times N \) target matrix, is defined similar to before, \( T = [t^1, \ldots, t^N] \).
The targets are approximated using,
\[ \hat{t}^n = W^T x^n \]
and \( W \) is now a \( d + 1 \times C \) matrix, estimated using the squared error and the pseudo inverse,
\[ W = (XX^T)^{-1} XT^T. \]
A new point is classified by looking at the estimated target. The position of the largest class entry is set as the class,
\[ \hat{c} = k, \text{ if } \hat{t}_k > \hat{t}_i, \forall i \neq k. \]
In figure 2.3 an extra class, noise, is included and linear classification using the described method is applied. The decision boundaries are well specified and no ambiguities, except on the boundaries, exist.

Because of the linearity of the problem the columns of \( W \), which we will call \( w_c \), can be found by solving the pseudo inverse for one class at a time. This equals a one-against-all approach, because each row of \( T \) is encoded in this way. In the two-class case the second row is equal to 1 minus the first row, \( t_2^n = 1 - t_1^n \), and therefore the second column of \( W \) is also a linear transformation of the first column, \( w_2 = [1 - w_{01}, -w_{11}, \ldots, -w_{d1}]^T \). The criteria for selecting class one is
\[
\begin{align*}
   w_1^T x & > w_2^T x, \\
   w_1^T x & > 1 - w_{01} + \hat{w}_1^T \tilde{x}, \\
   w_1^T x & > \frac{1}{2},
\end{align*}
\]
which can be seen to be a simple transformation of the $\pm 1$ criteria used in the two-class problem. This shows that the two methods are consistent with each other, and that the 1-of-$C$ encoding is a natural extension to multiple classes.

### 2.1.2 Skewed class sizes

If the number of samples in each class are not equal, it will influence the estimation of the weights. If the sample sizes reflect the expected rates of emergence in the prediction cases, nothing should be done. If you do not expect one class to appear more often than the other and thus would like them to have equal chances before seeing a data point, the error function should be changed slightly,

$$\mathcal{E} = \sum_n \lambda^n (\hat{t}^n - t^n)^2,$$

where $\lambda^n = \frac{1}{N_{cn}}$, and $N_{cn}$ is the sample size of the class that the $n$’th data point belongs to. The solution to this equation becomes

$$\mathbf{w} = (\mathbf{X}\Lambda\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{A}\mathbf{T}^T,$$

where $\Lambda = \text{diag}[\lambda^1, \ldots, \lambda^N]$. 

---

**Figure 2.4:** Linear classification with different class sample sizes can result in skewed results. If the different sample sizes reflect the expected sample sizes this is not a problem, but if you expect equal sample sizes in the prediction stage, you need to rebalance the training.
2.2 Artificial neural network

The brain consists of a very large number of neurons. Simplified, a neuron can be described as a unit that accepts a number of inputs and has only one output, which can be excited. The neurons are structured in a complex network, thus inputs to a neuron can be the outputs of other neurons. Using this complex and large network of simpler units the brain is able to process very complex tasks.

The artificial neural network, also called a multilayer perceptron [14], was inspired by the structure of the brain and an example is shown in figure 2.5. The input layer consists of \( x_1 \) to \( x_d \), the output layer is \( y_k \) and the layer in between is called the hidden layer. The layer between \( y_k \) and the targets \( t_k \) is called postprocessing. The hidden layer can have any number of levels and each level can have an arbitrary number of hidden units, the neurons. The network in figure 2.5 has one hidden layer.

The input to hidden unit \( j \) is named \( u_j^n \) and is given as the sum of the products
Classification

of the input weights, \( w_{ij} \) and the inputs, \( x_i \),

\[ u^m_j = w^m_{0j} + \sum_i w^m_{ij} x_i. \]

The outputs of the hidden units are given as,

\[ h^m_j = g(u^m_j), \]

where \( g() \) is a so-called activation function. The output, \( y^m_k \), is given as the sum of the products of the output weights and the outputs of the hidden layer,

\[ y^m_k = w^m_{0k} + \sum_j w^m_{jk} h^m_j. \]

The activation function is a function that mimics the excitation from the neurons in the brain. Functions that have been used include the step function, a linear function and the sigmoid function. If the linear activation function is used everything becomes linear and the complete network is reduced to a variant of the linear classifier from the previous section. The step function will divide the input space into regions and specify an output value for each region. Unfortunately the step function is not differentiable which makes training of the network hard. The sigmoid function provides a differentiable approximation of the step function. A variation of the sigmoid is the hyperbolic tangent which is centered around zero, and converts an infinite input space to the region \([-1; 1]\),

\[ h^m_j = \tanh(u^m_j). \]

The boxes with a 1 in the input and hidden layer are included to model offsets, which is necessary because the activation functions have a fixed center, zero in case of the hyperbolic tangent. The offsets are often included in the vectors, \( x_0 = 1 \) and \( h_0 = 1 \), like in the linear classifier, which allows the sums to be written as vector products,

\[ \begin{align*}
U &= W^{T}_{in} X, \\
Y &= W^{T}_{out} H,
\end{align*} \]

where \( U \) is the matrix of hidden unit inputs, \( u^m_j \), and likewise for \( X \leftrightarrow x^m_i \), \( Y \leftrightarrow y^m_k \) and \( H \leftrightarrow h^m_j \). \( W_{in} \) are the input weights, \( w_{ij} \), and \( W_{out} \) are the output weights, \( w_{jk} \).

The parameters of the model are the input weights and the output weights. These will be set using a training set, with input and outputs, and using an error function. The outputs of the neural network, \( y^m_k \), are linear combinations
of the outputs of the hidden layers and therefore can span the complete range in \( \mathbb{R} \). This is not useful for a classification task and postprocessing is needed of some kind. In the linear case this was done with a simple threshold of zero, or by comparing to the other estimates in the case of multiple classes. Both of these postprocessing procedures introduce non-differentiability to the error function, but a way of postprocessing suitable for classification is the softmax output function, 
\[
\hat{t}_k = \frac{e^{h_k}}{\sum_{k'} e^{h_{k'}}}.
\]
This function gives outputs in the range \([0; 1]\) which sums to one over the outputs and resembles the 1-of-\(C\) encoding of the targets. The targets can be interpreted as probabilities of each of the classes, and the most probable class would be chosen in the prediction stage. For training, the joint probability of the true classes is used and maximized as a function of the parameters. The probability of the true class can be found using
\[
p^n = \prod_k (\hat{t}_k^n)^{t_k^n}.
\]
when \( t_k \) is encoded using 1-of-\(C\), meaning that \( t_k = 1 \) if \( k \) is the true class, and \( t_k = 0 \) otherwise. Instead of maximizing the joint probability it is advantageous to minimize the negative logarithm of the joint probability, and an error function is found,
\[
\mathcal{E} = -\log \prod_n p^n = -\sum_n \sum_k t_k^n \log \hat{t}_k^n.
\]
The setting of the parameters can be done by differentiating the error function and using a gradient descent based procedure. Using the error back-propagation method \[76] the derivatives can be found to be
\[
\frac{\partial \mathcal{E}}{w_{jk}} = \sum_n h_j^n (\hat{t}_k^n - t_k^n),
\]
\[
\frac{\partial \mathcal{E}}{w_{ij}} = \sum_n \left(1 - (h_j^n)^2\right) x_i^n \sum_{k'} (\hat{t}_k^n - t_k^n) w_{jk'}.
\]
Look for details in appendix \[A\].

### 2.2.1 Complexity control

The number of inputs are usually set by the problem at hand, and so is the number of output classes, but the number of hidden units is not. You can choose any number of hidden units, and the more you choose the more complex functions can be modeled. On the other hand the more hidden units you choose the more noise the model is able to fit, and as always you have to choose the model size, i.e. the number of hidden units, with care. Three examples are
Figure 2.6: Five different settings of a neural network. The first three varies the number of hidden units and uses two, five and 50 hidden units. Clearly the complexity of the decision boundary increases, and it seems that the 50 hidden units fit noise in the data - you would not expect this to generalize well and the five units seems to be the best trade off. The last two figures have 50 hidden units, but different levels of weight decay. The fourth figure seems to be a good choice and the fifth has too much.

shown in figure 2.6. The first is too simple, the second seems right, and the third has too high complexity.

Another way of controlling complexity is by adding a part to the error function that penalizes large parameters. In this way you can model complex functions without modeling the noise,

$$\mathcal{E} = -\log \prod_{n} p^n = -\sum_{n} \sum_{k} t^n_k \log \hat{t}^n_k + \alpha_1 \sum_{ij} w^2_{ij} + \alpha_2 \sum_{jk} w^2_{jk}.$$

Using a squared error term, as in the equation above, is called weight decay because it decreases unimportant weights exponentially in a gradient descent algorithm. Cases with different amounts of weight decay is also shown in figure 2.6 where the case of (too) many hidden units can be forced not to fit the data tightly.

2.3 Kernel methods

The linear classifier has many nice attributes including simplicity, fast learning and convexity, but sometimes the modeling capability is too limited to fit the data properly, and as mentioned you can add functions of the inputs, the squares for example, to allow for more complex functions to be fitted. Extending the idea of adding inputs have been extended to possibly infinite feature spaces, in where the linear classification can separate any set of points. Obviously you cannot directly deal with an infinite feature space, but if you can define everything in terms of inner products in the feature space, these can sometimes be computed with finite computations [15].
Define $\phi(x)$ as the projection of the input, $x$, into a high dimensional (infinite) feature space. The output $t$ is modeled as a linear function,

$$ t = w^T \phi(x), $$

where $w$ is high dimensional. Using least squares, $w$ is found,

$$ E = \frac{1}{2} \sum_n (w^T \phi(x^n) - t^n)^2, $$

$$ = \frac{1}{2} w^T \left( \sum_n \phi(x^n)\phi(x^n)^T \right) w + \frac{1}{2} \sum_n (t^n)^2 - \sum_n w^T \phi(x^n)t^n, $$

and the solution becomes, $a = K^{\dagger}T$. In this equation only inner products of the projections are necessary, and for some projection schemes this can be computed in finite computations. $K = \Phi^T\Phi$ is called the Gram matrix and is a matrix of the inner products,

$$ K_{ij} = k(x^i, x^j) = \phi(x^i)^T \phi(x^j), $$

which is the essential part of the algorithm, and can be seen to be symmetric. Instead of performing an inner product of infinite dimensional vectors explicitly, you can compute the product for a range of feature spaces by only computing a function of the two input vectors of the original input space. The infinite space is not defined directly, instead you choose a kernel function, which implicitly selects the feature space. One such kernel function is the Gaussian kernel which is defined as

$$ k(x, x') = e^{-\frac{\|x-x\|^2}{2\sigma^2}}. $$

A new target is predicted like this

$$ \hat{t}^* = w^T \phi(x^*) = a^T \Phi^T \phi(x^*) = a^T k(x^*), $$

where $k(x^*) = [k(x^*, x^1), \ldots, k(x^*, x^N)]^T$ is a vector of the vector products between the new point, $x^*$, and the training set points, $x^n$. 

**2.3 Kernel methods**

17
Figure 2.7: Three different settings of a gaussian prior is shown using variances of 0.01, 0.1 and 0.5. Notice the ability to model complex structures and still having a smooth decision boundary.

Often the Gram matrix does not have full rank, and therefore cannot be inverted as is necessary for the pseudo inverse. This can be helped by adding a penalizing term to the cost function similar to weight decay, which changes the pseudo inverse solution to $\mathbf{K}^\dagger = (\mathbf{K} \mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{K}$, with $\lambda$ being the weight decay parameter.

In figure 2.7 three examples of the Gaussian kernel is shown with different variances in the kernel function. The variance varies the area of effect of a given point and thereby the smoothness of the function. In the end the complexity of the permitted functions is limited for increasing variance.

2.3.1 Factor models in kernel space

Apart from doing classification in the kernel domain you can also do feature extraction and dimension reduction in the infinite feature space. A very used procedure is Principal Component Analysis (PCA) and it was extended to kernel PCA by [79]. PCA seeks to reduce the dimensionality while explaining as much variance of the data as possible. The solution is to use a space spanned by the eigenvectors with the largest eigenvalues as explained in chapter 3. An eigenvector of the sampled covariance matrix, $\hat{\Sigma}$, can be written like this,

$$\lambda \mathbf{v} = \hat{\Sigma} \mathbf{v} = \frac{1}{N} \sum_n \phi(x^n) \phi(x^n)^T \mathbf{v},$$

from where we can see that $\mathbf{v}$ is spanned by the $\phi(x^n)$ and thus can be written like this,

$$\mathbf{v} = \sum_n \alpha_n \phi(x) = \Phi \alpha.$$
If this is used in the equations you get,

\[ \lambda v = \Sigma v \leftrightarrow \]
\[ \lambda \Phi \alpha = \frac{1}{N} \Phi \Phi^T \Phi \alpha \leftrightarrow \]
\[ \lambda \Phi^T \Phi \alpha = \frac{1}{N} \Phi^T \Phi \Phi^T \Phi \alpha \leftrightarrow \]
\[ N \lambda K \alpha = KK \alpha, \]

for which relevant solutions can be found by solving \( N \lambda \alpha = K \alpha \), see \[79\], which gives the eigenvectors of the Gram matrix. A projection of a new input then becomes,

\[ v^T \phi(x) = \alpha^T \Phi^T \phi(x) = \alpha^T k(x). \]

Kernel PCA can also be formulated as finding projections to maximize the sum of variances of the individual dimensions which can be written like this,

\[
\begin{align*}
\text{maximize: } & \quad \text{Tr}\{V^T \Phi \Phi^T V\} \\
\text{subject to: } & \quad V^T V = I.
\end{align*}
\]

where \( V = [v_1, \ldots, v_d] \) is a projection matrix with \( d \) dimensions, and the solution is the \( d \) eigenvectors with the largest eigenvalues.

Other projection schemes exist. Kernel partial least squares \[75\] is a technique that takes into account the class labels when finding the projections. Instead of maximizing the variance of the data, projections that maximize the covariance between class labels and the data is sought. This can be written as,

\[
\begin{align*}
\text{maximize: } & \quad \text{Tr}\{V^T \Phi T^T U\} \\
\text{subject to: } & \quad V^T V = U^T U = I,
\end{align*}
\]

where \( T = [t^1, \ldots, t^N] \), and \( V \) and \( U \) are projection matrices of the input and labels respectively.

Kernel orthonormalized partial least squares (KOPLS) \[9\] maximizes the variance of the data projected onto the labels,

\[
\begin{align*}
\text{maximize: } & \quad \text{Tr}\{V^T \Phi T^T T^T \Phi^T V\} \\
\text{subject to: } & \quad V^T \Phi \Phi^T V = I.
\end{align*}
\]

As for kernel PCA we present the projection matrix using the basis of the training set \( V = \Phi A \), thus allowing a finite representation of the infinite dimensional matrix. This changes the optimization problem to,

\[
\begin{align*}
\text{maximize: } & \quad \text{Tr}\{A^T K_x K_x K_x A\} \\
\text{subject to: } & \quad A^T K_x K_x A = I,
\end{align*}
\]

where \( K_x = \Phi^T \Phi \) and \( K_t = T^T T \).
2.3.2 Reduced kernel orthonormalized partial least squares

As was stated earlier the kernel methods produce a matrix of $N \times N$, which can lead to very big systems that are hard or impossible to handle. In [9] a reduced KOPLS algorithm was suggested which allows for very big data sets. In KOPLS the solutions were specified as linear projections of the data set points. This made possible a finite representation of the directions in infinite dimensional space. Instead of using the whole data set for representing the projections, a subset can be used, $V = \Phi_R A$, where $\Phi_R$ is a subset of the full training set $\Phi$. Since representation and later classification is based on the projections, this alleviates the problem of computing and storing the $N \times N$ matrix, because only the projections using the subset $N_R \times N_R$ is stored. The optimization of the projection parameters is still done using the complete data set, and therefore this procedure is quite different from sub-sampling the training set, which is of course the easy way of reducing very big data sets. The optimization scheme now becomes,

$$\begin{align*}
\text{maximize: } & \text{Tr}\{A^T K_R K_t K_R^T A\} \\
\text{subject to: } & A^T K_R K_R^T A = I,
\end{align*}$$

where $K_R = \Phi_R^T \Phi$. As mentioned this limits the storage requirements of the algorithm, which speeds up computations and make some problems feasible which would have been infeasible before, or would have required sub-sampling of the data set. As mentioned $KK^T$ does often not have full rank, which makes inversion impossible. In rKOPLS, $N_R$ functions as a regularization parameter making $K_R^TK_R$ full rank.

2.4 Bayesian, generative classifier

Another approach to classification is to model the distribution of the classes individually. When a new point is introduced the distributions can be used to find the class that fits the point the best. This is optimal in the sense that if your assumptions are correct, the optimal choice of prediction is the class with the highest likelihood.

Unlike the previous methods introduced, the Bayesian classifier allows the generation of synthetic data from a fitted model. You distinguish between discriminative models who’s goal is to discriminate between classes, and generative models which can be used to generate synthetic data.

The performance of the Bayesian classifier relies on the precision of the fitted distributions. You can base your choice of distribution based on prior knowledge,
or you can choose a broad class of distributions which can fit many distributions.

### 2.4.1 Probability theory and Bayes Theorem

The probability of an event, $a$, is stated as $p(a)$. The joint probability that both event $a$ and $b$ is stated as $p(a, b)$. The joint probability can be split using

$$p(a, b) = p(a|b)p(b) = p(b|a)p(a). \quad (2.1)$$

$p(a|b)$ is called the conditional probability of $a$ given $b$. It is the probability of event $a$ knowing that event $b$ already happened. Using equation (2.1) the following can be found,

$$p(a|b)p(b) = p(b|a)p(a) \iff p(a|b) = \frac{p(b|a)p(a)}{p(b)}. \quad (2.2)$$

Equation (2.2) is called Bayes’ Theorem and is important because it "flips" the probabilities in the sense that it shows the relation between $p(a|b)$ and $p(b|a)$. A probability takes on values between zero and one by convention. When dealing with non-discrete values, you need to integrate the probability density functions (pdf’s) to obtain probabilities. The value of a pdf takes on values from zero to infinity. Since the integrals will be over the same (small) interval you usually neglect the integral and compare the value from the pdf’s directly and these values will in the following be called likelihoods, which can mean both a discrete probability and a value from a pdf.

The left hand side of the equation (2.2) is called the posterior likelihood. On the right hand side, in the numerator we have on the left the data likelihood and on the right the prior likelihood. In the denominator the model evidence is placed, which acts as a normalizing constant term. This term can be found by seeing that the left hand side must sum to one,

$$\sum_a p(a|b) = \sum_a \frac{p(b|a)p(a)}{p(b)} = 1.$$

The denominator is independent of $a$ and therefore this can be rewritten as,

$$\frac{1}{p(b)} \sum_a p(b|a)p(a) = 1 \iff p(b) = \sum_a p(b|a)p(a),$$

which shows that the denominator is equal to the sum of the possible numerators.
Classification

To illustrate the naming conventions and the applicability of Bayes Theorem the small example of speech/music classification will be used again. To transfer the ideas from this example to the Bayes Theorem, \(a = c\) is the class of either music or speech, and \(b = x\) is the two dimensional feature vector. The prior likelihood, \(p(c)\), is the likelihood of class \(c\) before (prior) the observation, and \(p(c|x)\) is the likelihood of the class after (posterior) the observation. You can say that the likelihood of the class has been updated using the information from the observation. \(p(x|c)\) is the data likelihood and is the likelihood of a data point given a class. In discriminative methods you model and optimize the posterior likelihood directly whereas in the generative models you model terms on the right hand side of equation 2.2, most importantly \(p(x|c)\) and \(p(c)\). \(p(x)\) works as a normalizing constant, and is independent of the \(c\). Because of the independence of \(c\), the normalizing constant is not important in the first applications, but becomes essential in chapter 3.

### 2.4.2 Modeling

The model in the Bayesian classifier consists of the data likelihood and the prior likelihood. Usually most attention is given to the data likelihood, and the prior is either non-informative, meaning it is flat giving the same probability to all events, or trivial based on the occurrence in the training set. The data likelihood is on the other hand very important. First a class of distributions must be chosen. This can either be a very specific class, such as the Gaussian distribution or the Laplace distribution, or it could be a more general distribution like the Gaussian Mixture Model.

When a distribution class has been selected, the model can be parameterized and the posterior likelihood of the parameters can be written as,

\[
p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)},
\]

where \(\theta\) summarizes all the parameters of the model, and \(D\) summarizes the data, usually the collection of \(x^n\) and \(c^n\).

Depending on the level of inference, different approaches can be used to make a prediction. What you are basically interested in, is the probability of a class given the training set and a new measurement point, \(p(c^*|x^*, D)\). The joint probability of all variables is \(p(c^*, \theta, M, x^*, D)\), where \(M\) allows for more than one model to be used. The full Bayesian approach gives the complete distribu-
tion of \( c^* \) specified by the marginal likelihood,
\[
p(c^*|x^*, D) = \sum_{\mathcal{M}} \int p(c^*, \theta, \mathcal{M}|x^*, D) d\theta,
\]
\[
= \sum_{\mathcal{M}} \int p(c^*|\theta, \mathcal{M}, x^*, D)p(\theta|\mathcal{M}, x^*, D)p(\mathcal{M}|x^*, D)d\theta,
\]
\[
= \sum_{\mathcal{M}} \int p(c^*|\theta, \mathcal{M}, x^*)p(\theta|\mathcal{M}, D)p(\mathcal{M}|D)d\theta.
\]

The first term in the integral is the posterior likelihood of the class, where the dependence on \( D \) has been removed because for fixed \( \theta \), \( c^* \) is independent on \( D \). The second term is the posterior likelihood of the parameters and finally there is the posterior likelihood of the model. \( x^* \) has been removed from the last two likelihoods because they do not depend on it. Each of these posterior likelihoods can be evaluated using Bayes Theorem, and in the case of the posterior likelihood of the model marginalizing over \( \theta \).

The full Bayesian approach is quite involved and simpler approaches are widely used. Instead of marginalizing over the parameters often the most likely set of parameters are used instead, \( p(c^*|x^*, \theta_{\text{MAP}}, \mathcal{M}_{\text{MAP}}) \). This is done by maximizing the likelihood of the parameters by evaluating
\[
p(\theta|\mathcal{M}, D) = \frac{p(D|\theta, \mathcal{M})p(\theta|\mathcal{M})}{p(D|\mathcal{M})}.
\]

The denominator is constant compared to the parameters and only the numerator needs to be maximized. If the prior likelihood of the parameters, \( p(\theta|\mathcal{M}) \), is assumed to be equal for all values you only need to maximize the data likelihood, \( p(D|\theta, \mathcal{M}) \), and the result is called the maximum likelihood (ML) estimate. If you have an informative prior distribution of the parameters and maximize the joined likelihood of data and parameters you get the maximum a posteriori (MAP) estimate. It is called maximum a posteriori because maximizing the numerator is the same as maximizing the posterior likelihood because the denominator is constant.

The ML and MAP predictions can be related to the true Bayesian predictions by noting that if you instead of the posterior likelihoods of the parameters use a delta function on the ML or MAP estimate you get the same result. It is the same as saying that the ML and MAP predictions place complete confidence on the parameter estimates. In the limit of infinite samples the full Bayesian, ML and MAP converge to the same result.

In figure 2.8 two different classes of distributions have been fitted to the two classes using ML. In the figure to the left a single Gaussian has been fitted to
24 Classification

![Figure 2.8: To the left a single Gaussian with free covariance has been fitted to each class, resulting in a quadratic decision boundary. To the right a Gaussian Mixture Model has been fitted to each class, allowing much more detailed modeling of the data, and a more complex decision boundary.](image)

the data. In the right plot in the figure a Gaussian Mixture Model has been fitted to the data allowing a much more complex distribution to be fitted.

### 2.5 Probabilistic interpretations of classification models

The probabilistic interpretation of the Bayesian classifier is made explicit, but most other models can also be interpreted using probabilities, and there are links between special cases of the Bayesian classifier and other classification models.

In the two-class problem we can use a Gaussian distribution to model each class, and restrict the covariance matrix to be the same for both classes. Using Bayes Theorem the class with the largest posterior likelihood will be chosen. This can be evaluated by dividing the two likelihoods with each other, \( \lambda = \frac{p(c=1|x)}{p(c=2|x)} \) which is called the Bayes factor. If the Bayes factor is larger than one, class, \( c = 1 \), is chosen. The logarithm of the Bayes factor for the restricted Gaussians becomes,

\[
\log \lambda = \log \frac{p(c_1|x)}{p(c_2|x)} = \log p(x|c_1) - \log p(x|c_2) + k,
\]

\[
= (x - \mu_2)^T \Sigma^{-1} (x - \mu_2) - (x - \mu_1)^T \Sigma^{-1} (x - \mu_1) + k,
\]

\[
= w_0 + w^T x.
\]

This shows that the decision boundary is linear equivalent to the linear classifier.

Similarly it can be shown that if each class has its own covariance but they are
limited to be diagonal matrices the decision boundary becomes quadratic as if the squares of the inputs are added to the linear classifier. Finally using a full covariance matrix for each class has a decision boundary like a linear classifier including the squares and all the cross products of the inputs.

It should be noted that even though the decision boundaries share the same class, the different approaches are not exactly equal. For example, the number of parameters is often smaller in the discriminative (linear classifier) cases. Secondly, the optimization objective of a generative classification algorithm is not the same as for a discriminative and therefore in general will give different solutions. The generative classifier can predict if an outlying observation is found, whereas the discriminative classifier can only give the classification result.

Models can often be specified in a probabilistic way having a deterministic part plus noise. The linear classifier in this case becomes,

$$ t = w^T x + \epsilon. $$

If the noise is modeled using a Gaussian distribution the negative log likelihood error function becomes,

$$ \mathcal{L} = -\sum_n \log p(t^n|x^n) = -\sum_n \log \mathcal{N}(t^n|w^T x^n, \sigma^2), $$

$$ = \frac{1}{\sigma^2} \sum_n (t^n - w^T x^n)^2 + k, $$

which is very similar to the quadratic error function and have the same minimum.

The probabilistic interpretation of the 1-of-$C$ encoding was used in the post-processing of the artificial neural networks. The targets modeled the posterior likelihood of the classes, and the error function was the combined likelihood of the targets. A similar strategy can be used with the linear classifier using logistic postprocessing. This might work better in some cases, but you lose the linearity of the model and it has to be trained using an iterative algorithm.

## 2.6 Summary

This chapter has described the basic classification algorithms that was used in the project. These were an integral part of the audio classification projects of chapter 6 but was also used indirectly in the classification of parameters in chapter 7. The linear classifier is the basic approach which is fast and convenient.
in both application and interpretation. The artificial neural networks and the kernel methods present models with much broader capabilities, but comes with a complexity price which makes training and interpretation harder. The Bayesian framework presents a way of describing the methods with a joined formulation, as well as introduce some methods on its own.
Chapter 3

Independent Component Analysis

An important problem in audio analysis is the cocktail party problem. It consists of a number of sources (people speaking simultaneously), and a number of observations (microphone recordings). The sources will be present with different weights in each recording and we wish to obtain the original sources. If we neglect room acoustics and delays the problem can be described like this,

\[ X = AS \Leftrightarrow WX = S, \]

where \( X \) are the observations, \( S \) are the sources, and \( A = W^{-1} \) are the mixing weights. The only observed variable is \( X \) and we wish to obtain \( S \). An arbitrary invertible mixing matrix will be a solution and of course we need additional information or assumptions.

In Principal Component Analysis (PCA), the assumption is that the sources are uncorrelated. Unfortunately this does not give a unique solution since any rotation of the solution will give another uncorrelated solution that explains the data equally well. With the assumption of non-Gaussian and independent sources, a unique solution can be found. Two algorithms will be presented that solves the problem, the maximum likelihood approach and FastICA [42].

The method of convolutive independent component analysis (cICA) extends the
ICA by taking filtering into account,

\[ x^n = \sum_k A_k s^{n-k}. \]

Even though the method has not been used in this project, it would be odd talking about ICA and audio and not mentioning cICA. The reason is that in most circumstances recording audio in multisource/multirecording environments the sound paths from source to microphone will be too large to ignore the delay and the reverberation of the room. These two factors is exactly what cICA includes. Unfortunately it also makes the problem much harder to solve. For a comprehensive survey of cICA algorithms, see [70].

### 3.1 Principal Component Analysis

Principal Component Analysis (PCA) is a well known method of decomposing or projecting a data set. The method finds dimensions in the data that are orthogonal to each other and sorts the dimensions in order of explained variance, so that the dimensionality can be reduced while preserving as much variance as possible.

A number of observations of dimension \(d\) is given and could be the example of pitch features from chapter 2. The data set is \(X = [x^1, \ldots, x^N]\), and \(x^n\) is the \(d\)-dimensional observation. We want to find a projection possibly onto a lower dimensional space that makes the dimensions of the data uncorrelated to each other, and that maximizes the preserved variance. First a projection of the data onto a single dimension is sought which maximizes the variance of the data. The projection vector is called \(v\) and we want to maximize,

\[
E\{(x^T v)^2\} = E\{v^T x^n(x^n)^T v\} = v^T E\{x^n(x^n)^T\} v, \]

where \(\Sigma\) is the covariance of the data, which is assumed to be zero mean. Of course the relation can be maximized arbitrarily by increasing the length of \(v\). This is obviously not what we are interested in, and therefore the vector is constrained to have unit length. The maximization can be solved using a Lagrange multiplier,

\[
\frac{\partial}{\partial v} \left( v^T \Sigma v + \lambda (1 - v^T v) \right) = 2(\Sigma v - \lambda v) = 0, \Rightarrow \Sigma v = \lambda v.
\]
3.2 Independent Component Analysis

This is an eigenvalue problem, and the solution is the eigenvector and \( \lambda \) becomes the eigenvalue. A number of eigenvectors exist. And we want the one that maximizes the variance of the projected data. The variance can be found,

\[
\mathbf{v}^T \Sigma \mathbf{v} = \mathbf{v}^T \lambda \mathbf{v} = \lambda.
\]

which shows that the variance is given by the eigenvalues. To maximize the variance, we choose the eigenvector with the largest eigenvalue. This can be generalized to more dimensions, and to maximize the variance of any number of projections that are orthogonal to each other, you need to choose the eigenvectors with the largest eigenvalues. These projections are called principal components and the method is therefore called Principal Component Analysis.

If all dimensions are retained the projection becomes a rotation of the data. If you afterwards divide each dimension with the corresponding \( \sqrt{\lambda} \) you obtain a data set that has a covariance equal to the identity matrix, and the procedure is called whitening. If the data is not zero mean, whitening includes extracting the mean as well.

In general you do not know the covariance of the data beforehand, but a good approximation can be found using the sampled covariance,

\[
\hat{\Sigma} = \frac{1}{N-1} \sum_n \mathbf{x}^n (\mathbf{x}^n)^T.
\]

3.2 Independent Component Analysis

Instead of uncorrelatedness which was the criterium of separation in PCA, statistical independence is the criterium of Independent Component Analysis (ICA). The problem with PCA is that the optimal solution is not unique in the sense that other solutions exist which are as uncorrelated as the one found. This can be seen by looking at a solution and inserting an orthogonal rotation matrix, \( \mathbf{Q} \),

\[
\mathbf{X} = \mathbf{A} \mathbf{S} = \mathbf{A} \mathbf{Q}^{-1} \mathbf{Q} \mathbf{S},
\]

which can be interpreted as a new solution with mixing matrix \( \mathbf{A} = \mathbf{A} \mathbf{Q}^{-1} \) and sources \( \mathbf{S} = \mathbf{Q} \mathbf{S} \). If we look at the covariance of \( \mathbf{S} \),

\[
E\{\mathbf{S} \mathbf{S}^T\} = E\{\mathbf{Q} \mathbf{S} \mathbf{S}^T \mathbf{Q}^T\} = \mathbf{Q} E\{\mathbf{S} \mathbf{S}^T\} \mathbf{Q}^T = \mathbf{Q} \mathbf{Q}^T,
\]

we see that any rotational matrix \( \mathbf{Q} \mathbf{Q}^T = \mathbf{I} \) will result in the same covariance and therefore cannot be distinguished from the first solution. This is of course a problem in terms of interpretability of the sources, and for example with signal separation you will most likely get a mixture of the original signals.
Figure 3.1: Here three scatterplots of differently distributed data are shown. The first plot is Gaussian distributed, the second is an exponentiated Gaussian using an exponent of 3, and the last uses exponent 0.5. This generates distributions that are super-Gaussian and sub-Gaussian respectively. All of them have a covariance of the identity matrix. Clearly the second and third plots do not have independent dimensions even though they are completely uncorrelated.

In PCA we use the second order moment to separate the sources, so an obvious extension is to use higher order moments to solve the problem. For Gaussian distributions this is not possible because all higher order moments are zero for that distribution. Therefore we require that the source signals are non-Gaussian. An illustration is shown in figure 3.1.

Two general approaches to ICA will be presented here. The first is the maximum likelihood approach of which two variations, a fixed distribution and a parametric one, will be shown. The second is the FastICA method that does not assume a specific distribution. FastICA has the advantage of being quite robust and fast, where as the maximum likelihood approach has the advantage of allowing for further tools within the probabilistic framework. Other methods exist as well, especially the JADE algorithm has been popular, and for general treatment of the topic see [21], and for general treatment of the topic see [43, 48].

A completely unique solution cannot be found even when using ICA. As was shown, any rotation would result in the same covariance and in ICA this is limited to some special cases of rotations. Any scaling of the sources can be completely compensated for by scaling the columns of the mixing matrix appropriately. Since ICA in general does not specify a variance of the sources this results in a scaling ambiguity in ICA. Similarly, the labeling of the sources is not specified and therefore a permutation of the sources can be compensated for by permuting the columns of the mixing matrix, resulting in a permutation ambiguity. None of these ambiguities pose problems in ordinary use, since the source distributions are usually fixed to unity and the permutation rarely matters.
3.2 Independent Component Analysis

3.2.1 Maximum likelihood

Separation of the sources will be done by maximizing the likelihood of the data \([71]\), which is motivated in Bayes Theorem to produce the most likely parameters of the model given the data. The model of the data is the linear relation between observations and sources, \(X = AS\), and the distribution of the sources, \(p(s) = \prod_i s_i\). Using these relations the likelihood of the data, \(p(x^n)\), which we want to maximize can be written as,

\[
p(x^n) = \int\delta(x^n - As^n)p(s^n)ds^n.
\]

By a change of variable, \(z = As^n\), \(\frac{\partial s^n}{\partial z} = W\), this can be written as,

\[
p(x^n) = \int \delta(x^n - z)p(Wz)|W|dz, = p(Wx^n)|W| = |W|p(s^n).
\]

Since the observations are considered to be independently drawn from the same distribution (i.i.d.) the probability of the complete data set can be written as,

\[
p(X) = \prod_n p(x^n) = |W|^N \prod_n p(s^n) = |W|^N \prod_i \prod_n p(s_i^n).
\]

By assuming a distribution of \(s^n\) this quantity can be optimized, usually using the negative logarithm converting the probability to the negative log likelihood error function,

\[
\mathcal{L} = -\log p(X) = -N \log |W| - \sum_n \sum_i \log p(s_i^n). \quad (3.1)
\]

Cardoso \([19]\) showed that maximum likelihood is equivalent to the infomax algorithm derived by Bell et al. \([12]\), that they both optimize the Kullback-Leibler divergence \([47]\) between the distribution of \(Wx\) and the hypothesized distribution of \(s\). If we look at the logarithm of the probability of the data set, \(X\), divided by the number of samples, it can be written like this,

\[
\frac{1}{N} \log p(X) = \frac{1}{N} \log \prod_n p(x^n) = \frac{1}{N} \sum_n \log p(x^n).
\]

This quantity is the sample average of the log likelihood and will converge to the expected value given enough samples,

\[
\frac{1}{N} \log p(X) \approx \int p^*(x) \log p(x)dx,
\]
where \( p^* \) is the true distribution of \( x \). This integral can be rewritten,

\[
\frac{1}{N} \log p(X) \approx \int p^*(x) \log \frac{p(x)}{p^*(x)} p^*(x) dx,
\]

\[
= -\int p^*(x) \log \frac{p^*(x)}{p(x)} dx + \int p^*(x) \log p^*(x) dx,
\]

\[
= -KL\{p^*|p(x)| - H(p^*),
\]

where KL is the Kullback-Leibler divergence and \( H \) denotes the differential entropy. Since \( H(p^*) \) is independent of \( W \) optimizing \( \log p(X) \) means optimizing the Kullback-Leibler divergence.

Optimization of equation 3.1 can be done using the gradient. Either you set it to zero or, if that is not possible, you can use an iterative algorithm based on gradient descent. The gradient of the negative log likelihood is

\[
\frac{\partial \mathcal{L}^n}{\partial W} = -W^{-T} - \gamma^n (x^n)^T,
\]

\[
\gamma^n = [\gamma^n_1, \ldots, \gamma^n_L]^T, \gamma^n_i = \frac{p'(s^n_i)}{p(s^n_i)},
\]

(3.2)

where \( W^{-T} = (W^T)^{-1} = (W^{-1})^T \) and \( p'(s) = \frac{\partial p(s)}{\partial s} \). Pedersen et al. [69] found that it makes a difference whether you optimize with regard to \( W \) or \( A \). Obviously the minima in error space are the same, but convergence properties are different. Further more, in this project we inspect the possibility of setting parameters of the mixing or separation matrix to zero, and in this case it is very important to be able to optimize in both ways, because zeros in the mixing matrix does not in general convert to zeros in the separation matrix and vice versa. The gradient using the mixing matrix is

\[
\frac{\partial \mathcal{L}^n}{\partial A} = A^{-T} + A^{-T} \gamma^n (s^n)^T = -W^T \frac{\partial \mathcal{L}^n}{\partial W} W^T,
\]

using the same definition of \( \gamma \) as in equation 3.2.

Whether a closed form solution can be found of course depends on the distribution function, but is in general not possible. One exception is the Gaussian distribution and even though it was stated previously that you should use non-Gaussian distributions we will try it here,

\[
p'(s) = \frac{\partial}{\partial s} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{1}{2\sigma^2}s^2},
\]

\[
= -\frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{1}{2\sigma^2}s^2} \frac{1}{\sigma^2}s = -p(s) \frac{1}{\sigma^2}s.
\]
If this is used in equation 3.2 you get,

\[
\frac{\partial L^n}{\partial W} = -W^{-T} - s^n(x^n)^T,
\]

\[
= -W^{-T} - Wx^n(x^n)^T, \Leftrightarrow
\]

\[
AA^T = x^n(x^n)^T,
\]

which can be solved using an eigenvalue problem and results in the PCA solution. We see that the Gaussian distribution cannot solve the ICA problem.

A commonly used distribution in ICA is the hyperbolic secant distribution, \( p(s) = \frac{1}{\pi \text{sech}(s)} \). The derivative using this distribution becomes,

\[
p'(s) = \frac{\partial}{\partial s} \left( \frac{1}{\pi \text{sech}(s)} \right) = -\frac{1}{\pi} \text{sech}(s) \tanh(s) = -p(s) \tanh(s).
\]

The gradient then becomes,

\[
\frac{\partial L^n}{\partial W} = -W^{-T} + \tanh(s)x^T,
\]

which cannot be solved in closed form, because \( s \) is a function of \( W \). Instead an iterative algorithm can be used. In [19] it was noted that a mismatch between the assumed source distribution and the true one does not have to be critical. If the distributions are not too different the mismatch will simply result in a scaling of the sources. In practice you can divide distributions in two classes, sub- and super-Gaussian, and if both assumed and true distribution belong to the same class the algorithm should work [20]. The hyperbolic secant method is a super-Gaussian distribution, and the super-Gaussian class covers many kinds of signals including many audio signals such as speech and music.

In the left plot of figure 3.2 the independent components of the super-Gaussian distributed scatterplot in figure 3.1 have been found and plotted.

### 3.2.2 Gaussian mixture ICA

The method describe in the previous subsection is dependent on a fixed distribution and this causes some limitations - it means that only sub- or super-Gaussian distributions can be separated at the same time. A method that dynamically switches between sub- and super-Gaussian distributions was developed by Lee et al. [49]. Another approach is to use a parametric source distribution that can fit both kinds of distributions and train its parameters as part of the ICA algorithm. Such a procedure is Independent Factor Analysis (IFA) [10]. A general
Figure 3.2: The same data as in figure 3.1 but here the ICA algorithm is used. The first two are using the maximum likelihood algorithm using the sech distribution. In the sub-Gaussian case the true independent components are not found because a wrong source distribution is used. The third plot uses the Gaussian Mixture Model for source distribution and the fourth plot uses FastICA. The last two methods can deal with any non-Gaussian distribution and solves the ICA problem. If you inspect the last two plots closely you can see how the dimensions have been flipped, which is a result of the permutation ambiguity of ICA.

procedure including additional noise terms is described, but also a square noiseless model, equalling the ICA model here, is described. IFA uses the Gaussian Mixture Model (GMM) to describe the sources,

\[
p(s) = \sum_k \pi_k \mathcal{N}(s|\mu_k, \sigma^2_k),
\]

where \(\pi_k\) is the probability of Gaussian component \(k\), and \(\mathcal{N}\) is the Gaussian probability distribution with mean, \(\mu\), and variance, \(\sigma^2\). The Gaussian distribution is also called the Normal distribution, hence \(\mathcal{N}\) is used.

The model is trained iteratively and interchangeably between the ICA model and the GMM until convergence. This procedure is called seesaw by Attias [10]. The ICA model is trained using the gradient which is found using,

\[
\gamma_i^n = \frac{p'(s^n_i)}{p(s^n_i)} = - \sum_k p(k|s^n_i) \frac{1}{\sigma_{ik}^2} (s^n_i - \mu_{ik}),
\]
and the GMM is trained using the E-M algorithm \[14\],

\[
p(k|s^n) = \pi_k \mathcal{N}(s^n | \mu_k, \sigma^2_k) \sum_{k'} \pi_{k'} \mathcal{N}(s^n | \mu_{k'}, \sigma^2_{k'})
\]

\[
N_k = \sum_n p(k|s^n),
\]

\[
\mu_k = \frac{1}{N_k} \sum_n p(k|s^n) s^n,
\]

\[
\sigma^2_k = \frac{1}{N_k} \sum_n p(k|s^n) (s^n - \mu_k)^2,
\]

\[
\pi_k = \frac{N_k}{N}.
\]

This algorithm can be used to solve for sub-Gaussian as well as super-Gaussian distributions as shown in figure 3.2.

### 3.2.3 Fast ICA

This algorithm was presented by Hyvärinen \[42\] and has quickly become a popular algorithm because of its speed, robustness and ability to handle both sub- and super-Gaussian source distributions simultaneously.

FastICA works by maximizing the non-Gaussianity of the sources, and the reasoning behind this goal will be explained next. Each observation is a linear combination of the sources, and according to the central limit theorem \[33\] the sum will be more Gaussian than any of the original sources. An estimated source will consist either of a sum of the true sources or a single true source. Since a sum of sources will be more Gaussian than a single source we can estimate the true sources by maximizing the non-Gaussianity of the data \[44\].

Different measures of how Gaussian a distribution is exist. Since all higher order cumulants are zero for the Gaussian distribution any of these could be used, kurtosis being the most common. A problem with kurtosis is that it is quite sensitive to outliers, and a measurement can depend on only a few samples. In FastICA another approach is taken. Differential entropy of a distribution is given by,

\[
H(s) = - \int p(s) \log p(s) ds.
\]

This quantity measures how random a distribution is and thereby how much information is gathered when a sample is observed. How much random is a
vague expression, but the flavor of it can be explained by an example. If a random variable can take two values, ±1, and takes the value +1 99% of the time, obviously you do not get very much information when a +1 is observed, and even though the signal is random, it is not very random. The Gaussian distribution is maximally random given the variance, and the differential entropy of a Gaussian variable is the largest possible. Therefore differential entropy can be used to measure Gaussianity. If the differential entropy of a Gaussian distribution of the same variance as $p(s)$ is extracted from the differential entropy of the sources you get the following,

$$J(s) = H(s_{\text{Gauss}}) - H(s),$$

which is called the negentropy [17, 24]. It is zero for a Gaussian variable and will take on positive values for other distributions, and therefore makes for a very good measure of Gaussianity. Unfortunately negentropy is very hard to measure by sampling, and requires some kind of approximation or detection of the distribution. A general approximation is,

$$J(s_i) = (E\{G(s_i)\} - E\{G(s_{\text{Gauss}})\})^2,$$

(3.3)

where different functions $G$ can be used [41]. $G(y) = y^4$ can be used which is equivalent to the kurtosis measure, but note that the square is used in equation 3.3 which makes FastICA able to optimize in both directions away from Gaussianity, leading to both sub- and super-Gaussian distributions.

FastICA uses a fixed point algorithm to optimize the objective function in equation 3.3 one direction at a time. First the data is whitened. In order to conserve the uncorrelatedness of the data, only a rotation is allowed to find the independent components, which means that the remaining part of the matrix is orthogonal, and the vectors in the matrix have unit length and are orthogonal to each other. The algorithm takes Newton steps to update the direction,

$$w^+ = E\{x g(w^T x)\} - E\{g'(w^T x)\}w,$$

where $g$ is the derivative of $G$ and $g'$ the derivative of $g$. $w^+$ is not normalized to have unit length and is not orthogonal to other previously found directions. First the vector is normalized by dividing by the length of the vector, $(w^+)^T w^+$. Secondly the vector is orthogonalized using the Gram-Schmidt procedure [84]. An approach that finds the full mixing matrix instead of the sequential procedure also exists. It uses orthogonalization using,

$$W = (W^+(W^+)^T)^{-\frac{1}{2}} W^+.$$

In the right plot of figure 3.2 the independent components of the sub-Gaussian distribution in figure 3.1 have been found. Remark that the directions in the data have been aligned with the axis, and that no dependence is visible in the data.
3.3 Summary

In this section Independent Component Analysis was described in relation to Principal Component Analysis. Two very different approaches was described, maximum likelihood and FastICA. FastICA is becoming the method of choice, but the maximum likelihood approach describes the model in a probabilistic framework which allows model selection methods (chapter 4) to be used as in chapter 7.
This chapter will present different methods of how to choose between the models presented in the previous chapters. The methods of comparison consider different versions of the same model. An example could be the choice of how many hidden units you should use in an artificial neural network. Another could be how many inputs, and possible preprocessings of the inputs, to include in a linear classifier. This chapter deals with the choice complexity of nested models. Nested models means models for which a simpler model is contained in a more complex model. For example if setting the weight in a linear classifier to zero is equivalent to not including that parameter, thus setting the parameter to zero makes a more complex model equal to a simpler model, which does not include that particular parameter. The general procedure of evaluating different models is much harder and typically involves a test set on which the models can be compared. The full Bayesian approach also has the ability to compare different models since the true likelihood of the data given the model is found.

A common understanding when dealing with modeling and noise in modeling is that the function we try to model is smooth and that the noise which is added is small and independently distributed. The independence assumption means that the noise cannot be smooth, because otherwise one noise sample would depend on the previous noise sample.
Figure 4.1: A data generating model consist of a linear part (the dashed line) and some added noise. Five points have been sampled (in blue) and based on these a polynomial must be fitted. Three different polynomials with the number of parameters indicated in the legend. Obviously the more parameters the better the fit, but another five points have been sampled (in red), and the higher order models fit these points badly. The two parameter model has the true order and the fitted line corresponds closely to the generating model.

4.1 Assessing classification performance

For many models the more simple models are contained in the more complex ones. Take for example a linear classifier with five inputs to be the simple model. If you add the squares of the first input as the sixth input you have a new model with an additional dimension. You can make this more complex model return the exact same results, if you set the parameter of the sixth dimension to zero. This means that the more complex model is capable of modeling the exact same functions as the simpler model, plus the additional functions when the parameters are not set to zero.

When optimizing parameters you force the parameters into values that minimize an error function and therefore when increasing the dimensionality of the model, you can always achieve at least the same score as the smaller model and possibly a lower score. If the objective is to minimize the error you would prefer a very big model. A question is then if you always want to have as big a model as possible? Apart from the obvious argument of tractability, also from a classification performance point of view you do not want this, as will be explained with a small example.

In a classification task you are not interested in classifying the points in the
training set, because you already have the outcomes of these. Rather you are interested in classifying a new point for which you do not know the class, and it must be the ability to classify a new point that is the performance measure of the model. Five points from a linear function with additive noise are given in figure 4.1. Different models are trained on the training samples, and are in the figure as well. The squared error becomes smaller for increasing dimensionalities of the model, and reaches zero for the largest model. The test error, the squared distance between the points in the test set and the model, is smallest for the model with two parameters which illustrates the point that model complexity cannot be decided based upon training error.

4.1.1 Occam’s razor

The principle is attributed to William of Occam, who was a monk who lived in the 14’th century. The principle states that ‘plurality should not be posited without necessity’ [39], which means that simpler explanations are preferred. A very illustrative example is the sequence \((0, 0); (1, 1); (2, 2); (3, ?)\) for which we want the model to predict the number \(?\). The obvious guess would be 3, using the model \(y = x\). Another option which explains the data just as good is \(y = \frac{4}{3}x - \frac{1}{2}x^2 + \frac{1}{6}x^3\) for which the guess would be 4. So why is it that we prefer the first explanation? The first solution is the simpler and uses only one parameter whereas the other model uses three parameters without gaining anything.

The principle can also be explained in a probabilistic fashion. The more parameters there are in the model, the smaller the likelihood of a specific set of values of the parameters will be. This means that the likelihood of a model becomes smaller the more parameters are included, given that the different models fit the data equally well. The most likely model will then be the model with fewest parameters.

When noise is present in the data, you will often increase the likelihood of the model when adding more and more parameters, because the model is able to fit the noise. Thus you do not have the easy choice of a number of models with increasing numbers of parameters that perform equally well. Instead you should consider if the increase in likelihood of a model justifies adding the parameter. Three methods of making this evaluation is presented, Bayes’ Information Criterium, Akaiikes Information Criterium and Automatic Relevance Determination.
4.2 Forward selection/backward elimination

When doing classification you often come up with a number of features extracted from the data set that could be relevant for the classification. As mentioned in section 4.1, you do not necessarily want all the features present in your model because this might lead to over fit, and some features may be more relevant than others. If you consider not using some of your inputs and want to find the optimal (sub)set of inputs to use, you end up with an exponential increasing number of model evaluations to perform. The number of possible combinations of $d$ inputs are, $2^d - 1$, if you do not evaluate the trivial solution of not using any inputs.

Obviously the exhaustive search is not possible for even small numbers of features and another approach needs to be taken. One such approach is to select the single best feature first by performing a comparison of each feature on its own. When the best feature is found it is evaluated in combination with all of the remaining features, one by one, and the best is selected. Now you have two, which again will be evaluated in combination with the remaining features, one by one. By this scheme you create a sequence of features in order of relevance, and each model will have one more feature than the previous one.

The number of comparisons using this procedure is quadratic in the number of features, $\frac{d(d+1)}{2} - 1$, which is much better than exponential. For ten features this limits the number of model evaluations from 1023 to 54, which of course is a very significant reduction and converts the problem of finding the best model from intractable to very tractable. Of course, the found combination of features is not necessarily the best, and the optimization is only approximate. In practice, though, it is often a good approximation. The procedure is called forward selection.

Another approach is to train the model with all the features. Then the model is evaluated with each feature taken out, one by one. The best is selected and the best model with all but one feature is found. The procedure is continued and a sequence is found in the opposite direction of the forward selection case. This procedure is called backward elimination, and ideally should give the same sequence as the forward selection case. This procedure has the same complexity as the other. You can perform both schemes to get an indication of how well an approximation they give of the exhaustive search. If they agree on the solution or there is very little difference with the errors found in each level, it is likely that that a good model have been found.

In figure 4.2 forward selection and backward elimination has been performed on the full 28 features of the audio data set from chapter 2. The two methods agree
4.2 Forward selection/backward elimination

Figure 4.2: Forward selection and backward elimination. The curves agrees to a certain extend, but it is also clear that the methods are approximative - otherwise they would have returned the same (optimal) result. The exhaustive search is also indicated and can be seen to be a lower bound of the two other algorithms and is quite close to forward selection. In the figure to the right is shown the mean square error of the same experiment. Forward selection and exhaustive search cover each other and are hard to separate. The expected decrease in error for increasing model dimension is clear.

An interesting observation in figure 4.2 is that the errors are not decreasing for increasing model orders. This seems to be in contrast to what was claimed earlier with nested models. The two are not in contradiction though, but is a result of the training error function not being the same as the reported error function. The linear model is used and training is based on the squared error function. The reported errors are misclassification rates which are easier to interpret, but are non-differentiable and therefore less useful for training. The plot to the right is the sum square error and it is obvious that the error is monotonically decreasing for increasing dimensions.
4.3 Saliency measures

Nonlinear models, like the artificial neural network, do not have direct optimization solutions and therefore use iterative schemes for finding the optimal values of the parameters. These schemes are slow compared to direct solutions and the optimal parameters take time to get. In the forward selection/backward elimination procedure the effort to evaluate the model, or even to calculate the error for a changed parameter, can be too time consuming and approximations can help.

In this section the pruning of parameters rather than feature selection will be the focus. Pruning is very similar to the backward elimination of features. For example for the linear classifier they are equal, since there is one parameter per feature, and setting a parameter to zero is the same as taking the associated feature out of the input set, since this input will not affect the outputs anymore. In other cases like in the artificial neural network there can be more parameters per input, and parameters exist in the model which cannot be linked directly to an input, and here the relation to feature selection is a bit more vague, but the same sequential procedure can be used. The backward elimination procedure could be used as is, simply by setting all the parameters to zero one by one, recomputing the error, and pruning away the parameter that hurts performance the least. For some models, like the artificial neural network, the computation of the error can be complex and time consuming and therefore we will seek an approximation of the error when a parameter is pruned.
In the left plot of figure 4.3 all the parameters but one are fixed at the minimum solution and the error function is plotted as a function of the remaining parameter. The optimal point of the remaining parameter can be seen as the minimum of the curve, and we are interested in the increase in error when the parameter is set to zero, i.e. when the error function crosses the y axis. One approach, like in backward elimination, is to simply set the parameter to zero and recompute the error. A Taylor expansion can be used to approximate the function and evaluate the error when the parameter is set to zero,

\[ E \approx E_0 + \Delta w^T \frac{\partial E}{\partial w} + \frac{1}{2} \Delta w^T H \Delta w + \ldots, \]

where \( \Delta x = (w - w_0) \), \( w_0 \) is the point around which the function is approximated (the minimum of the error function), and \( E_0 \) is the error in this point. \( H = \frac{\partial^2 E(w)}{\partial w \partial w^T} \) is the hessian matrix of the second derivatives.

The first derivative is often used in the optimization of the parameters and would usually already be given. The first derivative is zero, because the present point is a minimum and therefore the first order approximation is a horizontal line. The first order approximation is thus that the error is unchanged.

The second order Taylor approximation is also plotted in the figure, and this provides a useful approximation of the error if the parameter is set to zero. If a single parameter in \( w \) is set to zero, it is the same as if \( \Delta w \) is zero but for one parameter which is set to the negative of the optimal value, \( \Delta w_i = [0, \ldots, -w_{0,i}, \ldots, 0]^T \). The increase in error can then be found by putting this into equation (4.1)

\[ \Delta E_i = \Delta w_i^T \frac{\partial E}{\partial w} + \frac{1}{2} \Delta w_i^T H \Delta w_i + \ldots \]

The first derivative is zero, and ignoring higher order terms gives the optimal brain damage approximation of the increase in error by setting a parameter to zero. This is often referred to as the saliency of the parameter,

\[ \Delta E_i \approx S_i = \frac{1}{2} \Delta w_i^2 H_{ii}. \]

In the backward elimination scheme we did not only remove features, we also optimized the remaining parameters before we evaluated the error function. A two dimensional plot of the approximated error function is shown in the right plot of figure 4.3 as a function of two parameters. If \( w_1 \) is set to zero the value perpendicular to the \( w_2 \) axis is found. This is clearly not the minimum of the error function on the \( w_2 \) axis, and therefore a better estimate of the saliency can be found by retraining the remaining parameters, \( w_2 \) in the figure. Within
the second order approximation this retraining can be modeled as well [38]. Setting a weight to zero can be written as $\Delta w_i + w_i = e_i^T \Delta w + w_i = 0$, where $e_i = [0, \ldots, 1, \ldots, 0]^T$ is the unit weight vector, with the number 1 at the $i$'th place. Minimizing the error function disregarding higher order terms, assuming zero first order term and under the constraint $e_i^T \Delta w + w_i = 0$ leads to the so-called optimal brain surgeon saliency,

$$S_i = \frac{1}{2} \frac{w_i^2}{[H^{-1}]_{ii}}.$$ 

Details can be found in appendix B.

The hessian matrix can be rather complex to compute and at first glance it seems that the approximation might be more time consuming to compute than simply setting a parameter to zero and recomputing the error. The reason it is still advantageous to use the approximation is that the hessian matrix only needs to be computed once for all the parameters, and the saliency is trivial to compute afterwards. For the manual approach the error must be recomputed for each parameter, and therefore it is often cheaper to use the OBD or OBS approximation.

### 4.4 Bayes Information Criterium

If you select a model family to model your data, usually you still have to select the complexity of the model. As explained earlier for nested models, the error function cannot be used because the error will always decrease for increasing model orders, and this does usually not reflect the objective of the model. Other measures are needed that take generalization and Occam’s razor into account.

When you look at a model from a probabilistic point of view you specify the likelihood of the data as a function of the data given the parameters and, sometimes implicitly, the model, $p(D|\theta, M)$. To assess the performance of a model, the likelihood of the model given the data, but independent of the parameters, can be used, $p(M|D)$. Using Bayes Theorem you can rewrite the likelihood,

$$p(M|D) = \frac{p(D|M)p(M)}{p(D)}.$$ 

If you do not have prior knowledge about which model order to choose, $p(M)$ will be chosen to be non-informative, leaving only the data likelihood, $p(D|M)$, to be found. This can be done using the likelihood of the data given the parameters,

$$p(D|M) = \int p(D|\theta, M)p(\theta)d\theta.$$
4.4 Bayes Information Criterium

\( p(D|\theta, M) \) is the model of the data, which is usually specified directly. For some models the integral can be solved directly, for example if the Gaussian distribution is used for both data likelihood and prior. For many models the integral cannot be solved directly, so an approximation will be used instead.

The negative log likelihood can be approximated similar to what was done in the previous section using a Taylor expansion of the error function. The expansion is done around a minimum of the negative log likelihood, the maximum likelihood estimate, and therefore the first order derivative vanishes. If higher order terms are neglected the approximation becomes

\[
\mathcal{L} = -\log p(D|\theta, M) \cong \mathcal{L}_{ML} + \frac{1}{2} \Delta \theta^T \mathbf{H} \Delta \theta,
\]

where \( \mathcal{L}_{ML} \) is the value of the negative log likelihood in the maximum likelihood estimate, \( \Delta \theta = \theta - \theta_{ML} \), and \( \mathbf{H} = \frac{\partial^2 \mathcal{L}}{\partial \theta \partial \theta^T} \) is again the hessian matrix of the second order derivatives.

The likelihood of such an approximation becomes a Gaussian distribution. The integral of a Gaussian distribution can be evaluated directly and simply becomes the normalizing factor,

\[
-\log p(D|M) \cong \int e^{-\mathcal{L}_{ML} - \frac{1}{2} \Delta \theta^T \mathbf{H} \Delta \theta} d\Delta \theta,
\]

where \( d \) is the dimension of \( \theta \). Using a Gaussian distribution to approximate the normalization of a distribution is called the Laplace approximation \([52]\). The negative log likelihood consists of a sum over the samples in the data set, and therefore the hessian matrix will also be a sum over samples, which means that you can expect the hessian to increase linearly as a function of \( N \). You can write it as \( N \tilde{\mathbf{H}} \), where \( \tilde{\mathbf{H}} \) is the sample average of the hessian, and the determinant of the hessian can be written as \( N^d |\tilde{\mathbf{H}}| \). When you take the logarithm of the determinant of the hessian, \( N^d \) and \( |\tilde{\mathbf{H}}| \) separates and you can approximate the term by assuming that the \( N^d \) term will dominate both \( |\tilde{\mathbf{H}}| \) and the \( 2\pi \) term. If this is used in the equation, Bayes’ Information Criterium (BIC) \([80]\) is found,

\[
-2 \log p(D|M) \cong \text{BIC} = 2\mathcal{L}_{ML} + d \log N.
\]

This criterium has the form of the negative log likelihood plus a penalty term, which consists only of the dimension of the data and the model, \( N \) and \( d \) respectively. The model that has the smallest BIC value is the preferred model.

In figure 4.4 BIC is plotted together with the negative log likelihood. Forward selection has been used to find the included features. In contrast to figure 4.2 the
negative log likelihood is monotonically decreasing because the true likelihood is used both for training and the reported errors in the plot. The structure of BIC as the negative log likelihood plus a penalty term is obvious since it is larger than the negative log likelihood for all numbers of features. BIC is order consistent meaning that the minimum will approach the true model order with probability approaching one for sample size going towards infinity [46].

4.5 Akaike’s Information Criterium

Akaike’s Information Criterium (AIC) is quite similar to the BIC in form,

$$AIC = \mathcal{L}_{ML} + d,$$

but the derivation of it is quite different from BIC. The negative log likelihood is a biased estimate of the error of the model, because it is calculated on the same data as is used when finding the parameters of the model. AIC tries to approximate an unbiased error, which means the error computed on another data set, also called the generalization error [5].

The equation can be seen to penalize the dimension less than BIC, and this can be explained by seeing that the generalization error is not hurt much for a too large model, whereas it will be hurt a lot for too small a model. BIC tries to identify the most likely dimension of the model and does not seek to optimize the error. AIC is not order consistent, but since the objective of the criterium
is not the true order but rather generalization error, not being order consistent is a result of balancing the risk of underfitting versus overfitting the data \[46\].

### 4.6 Automatic Relevance Determination

Continuing the probabilistic treatment of the models we will now investigate further the use of prior distributions over the parameters, and especially the dynamic control of such prior distributions during training. You can view the model selection procedure as part of the training of the parameters, if you include the model selection by setting some of the parameters to zero. Using the saliency this was done explicitly, but it could also be done using a prior distribution over the parameters. This permits a form of complexity control that does not necessarily set parameters to zero, but limits the magnitudes of the parameters. In this case the prior distribution of the parameters plays the same role as the weight decay in the artificial neural network. Under certain choices of distribution they become exactly equal. We start by repeating Bayes’ Theorem,

\[
p(\theta|D, \alpha) = \frac{p(D|\theta)p(\theta|\alpha)}{p(D|\alpha)}.
\]  

(4.2)

In the following, the prior distribution, \(p(\theta|\alpha)\), will be chosen to be a Gaussian distribution centered around zero and \(\alpha = \frac{1}{\sigma^2}\) plays the role of precision. This makes it equivalent to squared weight decay. Like in figure 2.6 a large value of \(\alpha\) forces the parameters towards zero and a small value allows the training to fit the parameters more freely.

Specification of \(\alpha\) is in general dependent on the data, and therefore it makes sense to try to identify \(\alpha\) based on the data. Blindly optimizing the negative log likelihood in equation 4.2 does not work since this will simply result in \(\alpha = 0\). Instead we will optimize the likelihood of \(\alpha\),

\[
p(\alpha|D) = \frac{p(D|\alpha)p(\alpha)}{p(D)}.
\]

If \(p(\alpha)\) is assumed to be non-informative and as usual \(p(D)\) plays the role of normalization, only \(p(D|\alpha)\) needs to be found. It can be found as the normalization constant of equation 4.2 and is found by marginalization of the numerator,

\[
p(D|\alpha) = \int p(D|\theta)p(\theta|\alpha)d\theta.
\]

In some cases this integral can be solved directly, but in others this is not possible and the Laplace approximation is used. The negative log likelihood
that is expanded using the Taylor expansion is given as,
\[ \mathcal{L} = -\log p(D|\theta) - \log p(\theta|\alpha), \]
\[ = -\log p(D|\theta) - \frac{d}{2} \log \frac{\alpha}{2\pi} + \frac{\alpha}{2} \theta^T \theta. \]
where the Gaussian distribution is used for \( p(\theta|\alpha) \). Using the Laplace approximation the negative log likelihood of the data given \( \alpha \) becomes,
\[ -\log p(D|\alpha) \approx \mathcal{L}_{\text{MAP}} + \frac{1}{2} \log |H| - \frac{d}{2} \log 2\pi, \tag{4.3} \]
where \( \mathcal{L}_{\text{MAP}} \) is \( \mathcal{L} \) with \( \theta = \theta_{\text{MAP}} \), \( \theta_{\text{MAP}} \) is the maximum a posteriori estimate of the parameters, see section 2.4.2.

The gradient of the negative log likelihood can be found using the Gaussian assumption on \( \alpha \),
\[ \frac{\partial (-\log p(D|\alpha))}{\partial \alpha} = \frac{\partial \mathcal{L}_{\text{MAP}}}{\partial \alpha} + \frac{1}{2} \frac{\partial \log |H|}{\partial \alpha}, \]
\[ = \frac{1}{2} (\theta_{\text{MAP}})^T \theta_{\text{MAP}} - \frac{d}{2\alpha} + \frac{1}{2} \text{Tr}(H^{-1}) = 0 \Leftrightarrow \]
\[ \alpha_{\text{new}} = \frac{d - \alpha_{\text{old}} \text{Tr}(H^{-1})}{(\theta_{\text{MAP}})^T \theta_{\text{MAP}}}. \]

Since \( \alpha \) depends on the \( \theta_{\text{MAP}} \) and \( \theta_{\text{MAP}} \) depends on \( \alpha \) they are trained in an iterative and interchanging way. First the \( \theta_{\text{MAP}} \) is found for a given \( \alpha \), and then a single update of \( \alpha \) is done using the equation above. The gradient could be set equal to zero directly and \( \alpha \) found in one step, but this has shown to be too greedy \[51\]. As explained, \( \alpha \) and \( \theta_{\text{MAP}} \) depend on each other and setting \( \alpha \) to its optimal value for a given \( \theta_{\text{MAP}} \) locks the procedure.

Instead of having a single \( \alpha \) for all the parameters, it is possible to specify an \( \alpha \) for each parameter creating a vector \( \alpha = [\alpha_1, \ldots, \alpha_d]^T \). The update equation becomes,
\[ \alpha_{\text{new}} = \frac{1 - \alpha_{\text{old}} (\theta_{\text{MAP}}^{-1})_{ii}}{\theta_{\text{MAP}}^2}. \]

Having an accuracy parameter per \( \theta_i \) means that each parameter is controlled individually. For an unimportant parameter, \( \alpha_i \) will go to infinity, meaning that the parameter is forced to zero effectively eliminating that parameter. For parameters that are important to the modeling \( \alpha \) will tend to go to zero, to allow the parameter to be set freely. This method is called Automatic Relevance Determination (ARD) by Mackay \[51\].
In this chapter various methods for selecting between competing models were described. When doing classification, most often one has to choose between features or different complexities of the models. Care must be taken because of the possibilities of overfit of the data, and the methods presented here have been used in both chapter 6 and 7, where especially chapter 7.2 uses all of the methods described and compares their performance.
This chapter will deal with features and models that are specific to the audio domain. A number of general observations have been elucidated during the work, and the basis of those will be given in this chapter. The focus will be on analysis of the sound, and not synthesis, and thus all models might not be good for generating sounds, but are suited to understand the structure in audio and to extract information about the audio. Different entries to audio will be described. Especially frequency analysis is important and will be described in the first section, but also pitch has been an important topic of the research and will be presented in section two. In section three the mel cepstral coefficients will be presented as they are very important features when doing audio modeling. Lastly, environments involving multiple audio sources and multiple recordings will be described and the challenges in these situations will be explained.

5.1 Frequency analysis / spectrogram

A very important part of audio modeling is understanding frequency related issues. Sound consists of waves that propagates through the air. Waves are linked tightly to frequency and therefore sound is linked with frequency as well. When we speak of frequency, usually it is in the family of sinusoids, but frequency
simply means the rate at which something is repeated \[26\]. In the sinusoids it means how many times the same wave is repeated during a certain time period, usually seconds. It is not by coincidence that we relate frequency of sound with sinusoids though. If a single sinusoid is played it is perceived as a clear note, but if a single squared wave is played it is much more noisy. A cosine, a sinusoidal function, as a function of time, \( t \), is written like this,

\[
x(t) = A \cos(2\pi ft + \phi).
\]

The cosine can be completely described by its magnitude, \( A \), frequency, \( f \), and phase, \( \phi \). For increasing frequencies the pitch of the sound is perceived to be higher and higher.

More cosines can be added together to form more complex signals. In figure 5.1 three different signals is shown. The first part is a single cosine, and in the middle is shown three cosines added together. The base frequency is the frequency of the smallest frequency that divides all frequencies in the signal by an integer, and it is also the frequency with which the total signal is periodic. The third plot consists of ten cosines. The base frequency is the same for all three plots, and the plots show different numbers of frequencies of the infinite series that approximates a boxwave,

\[
x(t) = \frac{4}{\pi} \sum_{k=1}^{\infty} \frac{\cos \left( \frac{2\pi(2k-1)ft - \pi}{2k-1} \right)}{2k-1}.
\]

### 5.1.1 Fourier analysis

Thus far we have concentrated on the generation of sound based on sinusoids, but what is more interesting in the context of audio analysis is to decompose a given signal into sinusoids. Joseph Fourier claimed that any function of a

\[
\]
variable could be expanded in a series of sines. Even though not entirely correct, since exceptions do exist, he is still acknowledged by naming the field in his honor. The presentation here is restricted to sampled signals, which means they are only defined at discrete points on the time axis. In digital systems the magnitude axis is also discrete, but here it will be assumed that the accuracy is high enough for this to be neglected, and hence the signals are treated in the complete real or complex domain.

Any sampled signal of length, \( N \), can be decomposed into a sum of sinusoids in the following way,

\[
x(n) = \sum_{k=0}^{N-1} \tilde{x}_k e^{j2\pi \frac{kn}{N}},
\]

which is a sum of complex exponentials with frequencies specified by \( \frac{kn}{N} \), where \( k \) only takes integer values. A complex exponential can be rewritten in terms of sinusoids like this,

\[
e^{jn} = \cos(n) + j\sin(n) \quad \leftrightarrow \quad \cos(n) = \frac{1}{2}(e^{jn} + e^{-jn}).
\]

In equation 5.1 the frequencies are already specified, and the magnitude and phase is specified by the complex constant \( \tilde{x}_k \). A complex number can always be written like a magnitude and a phase,

\[
\tilde{x}_k = A_k e^{\phi_k},
\]

\[
\tilde{x}_k e^{jn} = A_k e^{\phi_k} e^{jn} = A_k e^{jn + \phi_k}.
\]

If this is set into equation 5.2 you see that magnitude and phase comes into the right places.

If a signal is given and you wish to find the decomposition, you need to specify the constants, \( \tilde{x}_k \), which can be found by the following formula,

\[
\tilde{x}_k = \frac{1}{N} \sum_{n=0}^{N-1} x(n)e^{-j2\pi \frac{kn}{N}}.
\]

The coefficients are called fourier coefficients and are the basis for the fourier analysis. A discrete signal is completely specified by its fourier coefficients, and a fourier transformation of a signal means the specification of the fourier coefficients. A signal is usually defined to exist in the time domain, because the signal is a function of time, i.e. it evolves over time. The fourier coefficients are said to exist in the frequency domain, because they are a function of the
frequency $\frac{k}{N}$. The transformation from time to frequency domain is a linear transformation, and can be written as,

$$\tilde{x} = Fx,$$

where $F$ is a transformation matrix with elements $F_{kn} = \frac{1}{N} e^{-j\frac{2\pi}{N} kn}$. $x = [x(0), \ldots, x(N-1)]^T$ and $\tilde{x} = [\tilde{x}_0, \ldots, \tilde{x}_{N-1}]^T$. This transformation can be seen as a basis change, and because the transformation is orthogonal it is simply a complex rotation of the data.

The transformation is called the Direct Fourier Transform (DFT) and as stated above $F$ is an $N \times N$ matrix which means that the transformation has complexity $O(N^2)$. An optimization has been found called the Fast Fourier Transform (FFT) that significantly reduces the complexity to $O(N \log N)$ which has made the transform much more convenient, [25, 36].

### 5.1.2 Spectrogram

In figure 5.2 an example is shown of the so-called spectrogram [6]. The spectrogram combines the time and frequency domains by splitting up the time domain in smaller windows of length $l$ and computing the DFT of each window. The
result is a matrix with time in one dimension and frequency in the other called the short time fourier transform (STFT). If the absolute is taken of the STFT you get the spectrogram which is visualized in the figure. The absolute of the DFT, one column of the STFT, is called a spectrum. Each pixel in the figure represents the magnitude of a certain frequency in a certain window,

$$X(n, k) = \frac{1}{N} \sum_{n' = 0}^{l-1} x(n' + n)e^{-j2\pi \frac{kn}{l}},$$

where \( l \) defines the length of a window. Both a plot of the complete signal in both time and frequency domain are plotted on the sides of the spectrogram. Clearly the time plot does not give any clear information about the frequency content of the audio and the frequency does not reveal the time information. Looking at the spectrogram you can see the points in time where onsets are present and you also see which frequencies are present.

The spectrogram is a very nice way of visualizing audio, but it also makes a great preprocessing step for machine learning tasks and is used in a wide range of applications. There are some different settings that need to be addressed which affect the characteristics of the spectrogram and may have an impact on performance of the subsequent methods.

**Sampling rate**

The sampling rate does not have an exclusive connection with the spectrogram or even the fourier transform, but is related to the sampling of continuous signals. Since all methods in this project are based on digital design, obviously all signals are sampled, but since most signals are audio signals their natural format is continuous. When sampling a signal you must be certain that it does not contain frequencies above half the sampling rate. Half the sampling rate is often referred to as the Nyquist rate after Harry Nyquist an electrical engineer at Bell Labs [66].

If a signal contains frequencies higher than the Nyquist rate, they will be mirrored into lower frequencies causing aliasing. The higher frequencies will simply be added to the lower frequencies. In certain applications this can be used to facilitate a smaller sampling rate than specified by the Nyquist rate and still get usable data, but in general it should be avoided. The Nyquist rate should also be obeyed when downsampling a signal, which is done by low pass filtering the signal prior to downsampling.
Window functions

Signals in general exist in time both before and after the window you are looking at. This means, that when you do the frequency transformation actually you are not performing a frequency transformation of the signal, but of a cut off part of the signal. You can model this cutting off part of the signal by multiplying the signal by a rectangular window that is one in the part you perform the transform on and zero elsewhere. A multiplication in the time domain becomes a convolution in the frequency domain and this is why you do not observe a perfect delta function for signals with only one frequency, but see spill over effects into neighboring frequency bins. This spill over is exactly the convolution of the delta function with the frequency representation of the window function.

Many window functions exist and a small selection is illustrated in figure 5.3. The functions in time domain look a bit arbitrary, but the functions in frequency domain look quite different and have different advantages. The rectangular window is the most simple window in time domain and is often used out of convenience. It has the advantage of having a narrow main lobe, but the side lobes have significantly higher magnitude than the two other window functions. The hanning window has the largest drop-off of the side lobes and the first side lobe of the hamming window has the smallest amplitude of the three. The optimal window would be a delta function which would mean that no energy was spilled over into the neighboring frequencies. This is not possible and in practice, when designing windows, the trade off is between the maximum magnitude of the side lobes and the total energy contained in the side lobes [2].
5.1 Frequency analysis / spectrogram

Time and frequency resolution

When creating a spectrogram a number of parameters affect the resolution of the spectrogram. There are different kinds of resolution when dealing with the fourier transform and this needs to be explained. The first approach to the resolution is simply the number of points in time and frequency, which means the dimensionality of the matrix $X(n, k)$. The resolution of the discrete fourier transform can be enhanced by what is called zero padding. Zero padding means appending zeros in the end of the signal. This has the effect of interpolating the spectrum that has the same size as the length of the signal. It is called interpolating because the signal is already completely specified by the smaller spectrum and the extra points can be found by using the points already in the small spectrum,

$$\hat{x}_M = F_{M \times N} x_N = F_{M \times N} F_{N \times N}^{-1} \hat{x}_N = \tilde{F}_{M \times N} \hat{x}_N.$$  

$M > N, \tilde{F}_{M \times N} = F_{M \times N} F_{N \times N}^{-1}$ performs interpolation between the $N$'th and the $M$'th spectrum. Since the interpolation involves a matrix product, it will usually be done using zero padding before frequency transforming and not by interpolation.

The resolution on the time dimension can be increased by overlapping the time windows, which in principle can be done until you only shift the windows by one sample and achieve the same time resolution in the spectrogram as in the original signal.

The other kind of resolution does not depend on the number of points in the spectrogram but depends on the length of the windows. The window functions presented earlier all depend on the length of the windows, and the longer the windows, the narrower the width of the lobes. As was stated, you wish the spectrum of the window to approximate a delta function which would make you prefer as long a window as possible. On the other hand, within one window there is no time resolution which means that in order to capture short changes in the signal you need the windows to be as small as possible. As is evident there is a trade off between time and frequency resolution.

This difference in resolution means that you can have a lot of points in the spectrogram, but that does not mean that close frequencies can be separated. It also means that even if you have time resolution of one sample, it does not mean that you can detect changes in frequency with accuracy of one sample.
5.2 Harmonic signals

Many sounds encountered in every day life consist of harmonic sounds. A harmonic signal is perceived by people as a single tone, but they do not in general consist only of a single frequency. In general they are a sum of sinusoids whose frequencies are linearly connected in the following way,

\[ x(t) = \sum_{k=1}^{K} A_k \cos(2\pi f_0 t + \phi_k). \]

\( f_0 \) is called the fundamental frequency or the pitch of the signal. The pitch term is a little controversial because it is related to the tone perceived by people, and this can be manipulated without changing the base frequency. In most cases they will coincide, and throughout the project, pitch will be defined as the base frequency.

The phase, \( \phi_k \), is often ignored when dealing with audio. You can easily see the difference between the waveforms if the signals are plotted, but the perceived difference is very hard to observe in most cases. This leaves only the amplitude and the fundamental frequency to be defined. Therefore you often model a harmonic sound by a product of the magnitudes and a sequence of spikes of height one specified by the pitch, see figure 5.4. The envelope specifies the magnitude of the spikes and the pitch specifies the distance between the spikes. In this way you have a nice specification that accounts for the harmonic structure in the sound. Many sounds can be attributed to this model. An example could be the voiced parts of speech, or single notes in music. Even music pieces often have a dominant harmonic part that can be identified.

The pitch of a sound is perceived as the note of the sound, which in general speech is low or high pitch of a sound. The envelope can a bit roughly be described as the thing that separates the sound of a violin and a trumpet playing the same note, often called the timbre of the sound, although this is rather simplified.

\[ \text{Pitch} \times \text{Envelope} = \text{Model} \]
5.2 Harmonic signals

5.2.1 Pitch estimation

The pitch is a great feature to base information extraction on, because the pitch is very robust to manipulations of the signal. An example is the narrow frequency range of a telephone line, which does not change the ability to find out if the person you are speaking with is male or female. The telephone severely limits the frequency range, and thus dampens both low and high frequency harmonics. Of course using the pitch as a feature depends on the possibility to measure the pitch of the signal. The model in figure 5.4 is obviously idealized and in practice pitch estimation can be difficult, especially if the same method should work in diverse settings.

Many methods exist with very different approaches to the problem. Two large groups of pitch estimation techniques can be identified, separated by the domains in which they work. The domains are the time domain and the frequency domain. A few of the methods will be presented in the following.

5.2.1.1 Frequency based solutions

The frequency based pitch estimators first use a Direct Fourier Transform (DFT) to compute the spectrum of the signal. This is usually done on a small window, but the window size should be chosen with care, since the pitch should be constant within the window which means small windows, but the main lobe should not be too large so that the harmonics overlap, which means large windows.

A quite simple and fast frequency based pitch estimator is the harmonic product spectrum (see e.g. [29]). After the spectrum is found, it is downsampled a number of times. The original spectrum is multiplied element-wise to the downsampled versions which results in the harmonic product spectrum,

$$HPS(k) = \prod_{k' = 1}^{K} |\tilde{x}(k'k)|.$$  

$K$ is a constant that defines the number of downsamplings used. If the harmonic product spectrum is plotted it will exhibit a clear peak at the pitch frequency, but also at multiples of the pitch frequency. Depending on the parameter $K$ and the envelope of the signal, the correct peak can be smaller than the peak at multiples of the pitch frequency.

The harmonic product spectrum is a very fast method, and the DFT is what takes time. It has a tendency to find multiples of the pitch as is shown in figure 5.5. Instead of choosing the maximum peak you could also choose the peak with
the minimum frequency. This would alleviate the problem of getting pitch estimates at multiples of the true pitch. This could cause erroneous measurements too, because signals might include noise that will cause a number of smaller peaks in the harmonic product spectrum.

Another approach is to model the spectrogram by a series of equally spaced bumps \[11\], which is called a pattern match algorithm. Each bump models the form of the main lobe of the window function. For a given pitch candidate you only need to compute the amplitudes of each harmonic. The model is then compared to the signal using squared error, and using line search different pitch candidates are compared and the best fitting pitch is chosen.

One problem with the method is that half the pitch will fit the spectrum at least as well as the true pitch, because if the magnitude of the bumps with odd indexes are set to zero, the model is exactly the same. If half the pitch is fitted it will result in a saw tooth like behavior of the magnitudes, which means a high second order derivative. Procedures that penalize the second order derivative can help alleviating the problem.

A combination of the harmonic product spectrum and the pattern match algorithm was proposed in \[62\]. It uses the feature that the harmonic product spectrum proposes a pitch of double the frequency and pattern match favors pitches of half the frequency. Combined you can find the true pitch and limit the number of comparisons in the line search of the pattern match algorithm.

If the log is taken of the spectrum the multiplication of the pitch and envelope in the model in figure \[5.4\] becomes a sum. If you take a second DFT of the log spectrum the transform also becomes a sum of the DFT of the pitch and the DFT of the envelope. According to the model the envelope should consist of low frequency parts, and because the pitch is periodic in the harmonics the
DFT of the pitch should reveal this structure. The DFT of the log spectrum is called the cepstrum and several algorithms have been proposed using it for pitch estimation, see, e.g., [31, 65, 77].

5.2.1.2 Time domain methods

The widest known method of finding the pitch is using the auto correlation function,

$$c(\lambda) = \sum_{n} x(n)x(n-\lambda).$$

For an ideal periodic signal the autocorrelation function of lag equal to the period of the signal will equal the autocorrelation function at zero lag. In less ideal situations peaks will occur at multiples of the pitch period, and the pitch period can be identified by selecting the peak with smallest lag [82]. In noisy conditions the peak can be hard to identify. Many methods have been proposed extending the autocorrelation method, for example the resolution has been enhanced [56, 89].

You can also model a signal as a harmonic part plus a noise term,

$$y = x + \epsilon.$$  

This allows for different kinds of probabilistic models of the data. Wise et al. proposed a maximum likelihood method [87], and a full Bayesian approach was taken by Hansen et al. [37]. A model is presented of the signal,

$$y(n) = \sum_{k} a_{2^k-1} \sin(2\pi k f_0 n) + a_{2^k} \cos(2\pi k f_0 n) + \epsilon.$$  

The reason for modeling the signal using both sin and cos is that you can omit the phases in this case, and control phase by adjusting the amplitudes, $a$. The phase is a circular quantity and harder to model than amplitudes, and the placement of the parameter in the model makes it even harder. The model can be expressed in matrix notation,

$$y = Xa + \epsilon,$$

where $K$ is the number of harmonics present in the signal. The likelihood for a given $f_0$ and a given $K$, is proportional to the likelihood of the data given these
Audio modeling

two parameters, \( p(K, f_0 | y) \propto p(y | K, f_0) \), and this quantity can be found by marginalizing the likelihood of the data given the model. The noise is modeled as zero mean Gaussian with variance \( \sigma^2 \), and the marginal likelihood becomes,

\[
p(y | K, f_0) = \int \int p(a, \sigma^2) p(y | K, f_0, a, \sigma^2) \, da \, d\sigma^2.
\]

Using conjugate priors \[52\] an analytical expression can be found and a grid search approach can find the most likely pitch. The number of harmonics can be marginalized too by applying a prior and summing, or the maximum can be found depending on application. You can also approximate the full Bayesian approach by using variational methods \[63\].

The MUSIC algorithm considers cross-correlations between the signal and delayed versions of the signal. Originally \[78\] the delayed versions were found using an array of microphones and relying on the finite speed of sound. Instead of the multiple microphones you can also consider the same signal using different numbers of sample-delayed versions of it. Unlike the original algorithm that searches for arbitrary signals, the model will be specified using a base frequency and multiples of it \[22\].

If the signal and the delayed versions are concatenated in a vector, \( \tilde{y} = [y(n), y(n-1), \ldots, y(n-(M-1))] \) it can be written like this,

\[
\tilde{y} = Xf + \epsilon,
\]

\[
X = [x_{\omega_0}, \ldots, x_{K\omega_0}],
\]

\[
x_{\omega} = [1, e^{-j\omega}, \ldots, e^{-j(M-1)\omega}]^T,
\]

\[
f = [a_1e^{j(\omega_0n+\phi_1)}, \ldots, a_Ke^{j(K\omega_0n+\phi_K)}]^T,
\]

where \( \omega = 2\pi f \). The covariance of this vector can be modeled like this,

\[
R = E\{\tilde{y}\tilde{y}^T\},
\]

\[
= XE\{ff^T\}X^T + \sigma^2I.
\]

\( E\{ff^T\} \) is diagonal and contains the squared amplitudes. If you assume that we use more delays than signals \( XE\{ff^T\}X^T \) will be singular and have \( K \) positive eigenvalues and the remaining will be zero. Thus, \( R \) will have \( K \) eigenvalues larger than \( \sigma^2 \) and the remaining will be \( \sigma^2 \).

The eigenvectors of the largest eigenvalues, those originating from \( E\{ff^T\} \), will span what we call the signal space, and the remaining eigenvectors span the noise space, \( G \). These two spaces can be found from the approximated \( R \) that is created using the audio signal. Then the matrix, \( X(\omega) \) is created and projected onto the noise space and if this is done for a range of frequencies a
The pseudo spectrum can be created using the following formula,

\[ P(\omega) = \frac{KM(M-K)}{\|X(\omega)^T G\|}. \]

The algorithm is extended to account for slightly inharmonic signals in [23], and also the extended version of the MUSIC algorithm called ESPRIT [67] has been used in pitch estimation, see, e.g., [31].

5.3 MFCCs

The mel frequency cepstral coefficients (MFCC) is a form of feature extraction and has been used in a wide range of areas. The MFCC originates in speech recognition [72], but has increasingly been used in other areas as well, such as music genre classification [55] [4], music/speech classification [59] [60] and other areas [88] [34]. Different variations exist, and the one described here is from [81].

First you take the DFT of a small window, as described in the first section of this chapter. Any window function can be used, but commonly the Hamming window is used. Next, the frequency axis is scaled using mel frequency filters. These are triangular filters, with center frequency scaled in a near exponential ordering,

\[ y(l) = \sum_k |\tilde{x}(k)| H(l, k), \]

where \( |\tilde{x}(k)| \) is the magnitude of the DFT, and \( H(l, k) \) is the mel frequency filter of coefficient \( l \). The logarithm is taken of each of the coefficients, and finally another DFT is done as if the \( y(l) \) were a signal on its own. The magnitude of the last DFT constitutes the MFCC. In figure 5.6 an example of the mel frequency spaced filterbanks is shown.

The MFCC’s are hard to analyze directly, but can be related to the cepstrum, which uses the same procedure as the MFCC but without the rescaling of the frequency axis: DFT \( \rightarrow \) log magnitude \( \rightarrow \) DFT \( \rightarrow \) magnitude. The cepstrum is simply a frequency analysis of the log spectrum, and the lower coefficients are related to the low-frequency content of the spectrum, which is closely related to the envelope of the signal. In harmonic signals the harmonics are equally spaced in the spectrum and therefore shows periodicity. This is caught by the cepstrum for higher coefficients. Therefore the cepstrum separates pitch and envelope information. This interpretation does not translate directly the MFCC’s, but they have shown a surprising versatility in areas of applications which by first glance
would seem like conflicting each other. For example MFFC’s have been used both in speech recognition and in speaker recognition. These two applications should be conflicting as speech recognition wants to be independent of speaker and speaker identification wants to be independent from the spoken content.

5.4 Multi speaker situations

Most of what have been described until now has focused on a single recording of a single source, but many times the picture is more complicated than this. For example many recordings include noise beside the signal of interest. Other times you can have multiple signals in the same recording. This could be in recordings of a conversation, where you will have two different people speaking, sometimes even at the same time. This of course makes analysis harder.

Another situation is that you can have multiple recordings of the same event. In this case you want to use the information from all recordings in your analysis. Especially if you have noise or multiple sources, having multiple recordings will help considerably, and leads to for example ICA algorithms as described in chapter [3] or for example beamforming algorithms [85].

One situation is of particular interest and is motivated by the emerging field of Lifelogs [28]. Lifelogs was originally presented by Bush [18] in a very foreseeing way. It contains the concept of recording every aspect of your (digital) life, including mails, webpages, documents... [35] [13] and as a still increasing part of everyday life becomes digitized, the Lifelogs become increasingly useful.

The most common forms of Lifelogs only record your life on the computer. This includes easy access to mails, visited websites, documents and so forth, and structuring it in a way that makes the data accessible. The advantage of such a
system is like taking notes from a conversation - you can always recall important information in an exact way. But the Lifelogs does not have to be limited to your life on the computer, and with the invention of wearable computers [53] with cameras and microphones a whole new world is opened. Audio Lifelogs is the audio of a person equipped with a microphone, which records all the time, recording every part of your life. This involves huge legal and personal aspects, but has some very useful points too. A great deal of information can be retrieved from audio using speech recognition [72] to facilitate search based on content [58], emotion is becoming a recognized property of speech [86] [61] and audio events is often used in audio classification. In [30] audio was used to reveal common audio stamps and thereby to structure a work day in groups, based only on recordings made by a small mp3 recorder. This way you would have an exact time stamp of when a door was closed or a window broken. Speaker recognition is another important point, which in conversations with many people would give you information of who had which opinions.

In this project the approach have been to merge multiple audio logs from many people and finding structure in this. This gives information about who was present at a given conversation, or at a given meeting, which again gives information about who received the information that was given on that meeting. This could prove essential in security announcements and other related topics.

Especially with audio, the problem becomes bigger when handling multiple recordings. The speed of sound is quite low compared to the audible range, which is very noticeable in large places such as valleys and stadiums, where the delay of reverberation or the delay of singing from one side to the other is in the order of tenths of a second or even more. Also the reverberation of most surfaces is too large to be ignored, which means that the recorded sound is not the source but a filtered version of the source. In a single microphone setup, this is not a big problem. Of course the filtering is as large as in multi microphone setups, but you only have a single version of each source, and you will never know that you observe a filtered version of the true source. In multi microphone setups the filtering is different for each recording. In microphone arrays the most prominent difference is the delay of the signal, and this can be modeled with relative ease, but it gets worse for arbitrary placement schemes as both delay and filtering can vary substantially.

5.5 Summary

In this chapter the foundation of audio modeling was given. A very important tool of audio discovery is the spectrogram which allows a very intuitive
visualization of the audio. The pitch is an important part of sections 6.1 and 6.2, whereas MFFC’s are used in section 6.3. The general setting of multiple speakers is investigated in chapter 7.
Chapter 6

Audio classification

6.1 Pitch based sound classification

PITCH BASED SOUND CLASSIFICATION

Andreas B. Nielsen, Lars K. Hansen
Intelligent Signal Processing,
IMM, DTU,
DK-2800 Lyngby

Ulrik Kjems
Oticon A/S,
Kongebakken 9,
DK-2765 Smørum

ABSTRACT
A sound classification model is presented that can classify signals into music, noise and speech. The model extracts the pitch of the signal using the harmonic product spectrum. Based on the pitch estimate and a pitch error measure, features are created and used in a probabilistic model with softmax output function. Both linear and quadratic inputs are used. The model is trained on 2 hours of sound and tested on publicly available data. A test classification error below 0.65 with 1 s classification windows is achieved. Further more it is shown that linear input performs as well as a quadratic, and that even though classification gets marginally better, not much is achieved by increasing the window size beyond 1 s.

1. INTRODUCTION
The classification of sound is important for hearing aids because different sound environments, like music and speech, require different processing to get the optimal experience for the user. Hearing aids rely more and more on automatic program selection, which requires accurate sound environment classification.

The pitch is an interesting feature of sound and is used in various situations. One of the reasons pitch is interesting is that it is somewhat immune to frequency modulation like for example low pass filtering. If for example sound is transferred through a telephone, the pitch of the signal remains the same, and it is only the estimation of the pitch that is affected. Potentially this can make the classification more robust.

For a short survey of pitch estimation references look in [5]. In [2] the pitch is used for blind source separation, music information retrieval based on pitch is done in [7] and in [12] the pitch is used for language identification. The pitch has also been used for classification. In [4], [15] and [16] the pitch mean, variance and difference are used as features.

In this paper a classification system based solely on the pitch will be created. We are interested in three classes, broadly defined as; music, noise and speech. For such a system a pitch estimator, pitch features and a classification model is necessary. For running in a hearing aid it is important to have simple algorithms. The pitch estimation can be the most demanding part of the system, but in this paper the quite fast harmonic product spectrum will be used. Again to enhance efficiency, effort goes into finding features that separates the classes well, instead of using a complex classification model. A simple probabilistic model is used for the actual classification. To evaluate the system, it will be trained using a collection of sound that captures many of the aspects of both music and noise. A test set of publicly available sounds has been gathered to facilitate comparisons.

In the second section the pitch estimation, the pitch features and the classification model will be presented. In section three the data used for both training and test is described. Section four contains the results followed by the discussion.

2. METHODS
In this section the pitch estimator is presented. A selection of the investigated features are described and finally the classification model is presented.

2.1. Pitch estimation
The harmonic product spectrum (HPS), see e.g. [6], exploits a simple feature of the frequency spectrum of a harmonic signal. When downsampling the spectrum the harmonics align with the fundamental frequency. When the original spectrum is multiplied with the downsampled spectra the HPS appears.

The pitch is estimated by taking the maximum of the HPS. The value $R$ defining the number of downsamplings is set to 5 in this paper.

In [13] a combined algorithm using the HPS was compared to a Bayesian pitch estimator [9] and HMUSIC [5]. The combined algorithm was superior for classification and the HPS shows similar performance on its own. The HPS has problems with doublings of the pitch, but this does not affect classification, because the pitch dynamics are not affected by doubling the pitch. The advantage of HPS is speed and lack of noise. Other algorithms have better accuracy, but tend to overlook frequencies if too small a search grid is used, which can be necessary to decrease computation time.

Apart from the pitch, a measure of the pitch error is found as well. This measure is called reliability, $r$. It is based on the relation between the energy in the maximum peak in HPS and the total energy in the HPS.

$$r = \frac{E_{\text{pitch}}}{E_{\text{HPS}}}$$

This measure is close to zero when there is very little pitch in the signal and close to unity when the pitch is well defined.

2.2. Features
Features are build on top of the two signals, $p$ and $r$. A total of 28 features are found, but only four will be described here. For the complete listing refer to [13]. An example from
6.1 Pitch based sound classification

Each class is shown in Figures 1, 2 and 3. In the captions of the figures the characteristic properties of the classes are described and illustrated.

Reliable windows are used to divide pitch samples into smaller subsets. The pitch samples in a reliable window are characterized by abs-difference values smaller than a threshold. The reliable windows in music capture a musical note each and in speech a voiced region, see Figures 1 and 2. Some of the features use the reliable windows and some features use the complete classification window.

Next, the four features that yielded the best performance in the forward selection procedure are described.

Fig. 1. Music. The pitch is confined in steps (A) which is caused by the musical notes. Note the very constant pitch in each step. In the bottom of the pitch plot is shown the reliable windows (B) (see text for explanation); each captures a note. The maximum of the reliability values is close to unity (C) and the minima are relatively high (D). This reflects the fact that pitch is dominant in music. Most dips occur between notes.

Fig. 2. Speech. Notice the long reliable windows and the changes of the pitch inside a window (A). The pitch is used to emphasize words in speech. Also notice the high maxima (B) and low minima (C) of the reliability. This reflects the differences between voiced and unvoiced regions, consisting of pitch and white noise respectively.

ReliabilityDev: This feature is the standard deviation of the reliability signal within the classification window,

\[ f_{\text{ReliabilityDev}} = \left( \frac{1}{I-1} \sum_{i=2}^{I} (r_i - \mu)^2 \right)^{1/2}, \]  

where \( I \) is the number of pitch samples in the classification window. This feature shows good separation of all classes with speech having high values, noise low and music in between.

Difference1: Because it is especially the dynamic properties of the pitch that describe the classes well, the difference between subsequent pitch samples is a good measure. The feature is based on a histogram of the pitch abs-difference values and is the number of values in the interval \([0;2]\). It is good for separating music from speech and noise, because music has many constant pitch parts and therefore a high Difference1 value.

ToneDistance: This feature is good for separating the music from speech and noise. It is the distance from the pitch to a 12'th octave musical note. First the pitch is converted to the tone scale, \( t \). This scale takes on integer values for notes and in between values for other pitches. The feature is the mean of the distances to the closest tones,

\[ t_i = 12 \log_2 \frac{p_i}{440}, \]  

\[ f_{\text{ToneDistance}} = \frac{1}{I} \sum_{i=1}^{I} |t_i - \text{round}(t_i)|. \]  

PitchChange: The PitchChange feature measures the difference between the highest and the lowest pitch in a reliable window and calculates the mean over a classification window,

\[ d_w = \max(p_w) - \min(p_w), \]  

\[ f_{\text{PitchChange}} = \frac{1}{W} \sum_{w=1}^{W} d_w, \]  

with \( W \) being the number of reliable windows, and \( p_w \) a vector of the pitch values in reliable window \( w \). A description of the remaining features can be found in [13].

2.3. Classification model

In this paper a probabilistic model is used based on the softmax output function [3]. The model is used because of its simplicity,

\[ z_c = \sum_{i=0}^{F} a_{i,c} f_i, \]  

with
Fig. 3. Noise from a café. The pitch estimate in noise is often random in nature, but is also constant at times, depending on the type of noise. The reliable windows are generally short with little change of the pitch and do not hit musical notes. The reliability of noise is often low and with low variation compared to music and speech.

where \( F \) is the number of features, \( f_i \) are the feature values, \( f_0 = 1 \) is the bias and \( a_{i,c} \) are the variables of the model. In order to get the desired output interval of 0 to 1, the softmax output function is used,

\[
y_c = \frac{\exp(z_c)}{\sum_{c'} \exp(z_{c'})},
\]

for \( C \) classes. This output function also assures that the outputs sum to 1 and they can be interpreted as probabilities.

Three variations of the input are used. The linear with the features used directly (Linear). A quadratic including the squares of the features, thus doubling the input size (QuadDiag), and last a quadratic where all the covariance combinations are used making a total of \( i(i + 3)/2 \) inputs (excluding the bias) to the model (QuadComp).

3. DATA

The database used for training the model consists of the clean classes of music, noise and speech. The music has been taken from various recordings from a variety of genres totalling, 50 minutes. The noise consists of traffic, factory noise, a helicopter, many people talking and various other sources, totalling 40 minutes. The speech was taken partly from the Keele [11] and KED Timit [10] clean speech databases and from other clean speech sources in different languages, totalling 42 minutes. A complete description can be found in [13]. For the test set publically available sounds were used. Eight music files totalling 38 minutes, 23 minutes of noise, and 35 minutes of speech were included. The specific files can be found here [14].

The pitch estimator uses windows of 100 ms and overlap of 75 ms. This makes approximately 40 pitch samples per second. The classification windows of above 1 s uses overlap to obtain a classification every second. With these settings the training set size is approximately 7000 samples and the test data is approximately 5500 samples.

4. RESULTS

In total 28 features were found. Forward selection is used to select between them. Results with 1 s classification windows is shown in Figures 5 and 6.

The three plots of the test errors of Figure 5 shows no improvement when using more than 7 features. The more complex models show better training error, but when it comes to test error not much is gained, and from five features the linear model performs better. This is verified in Figure 6 where the three models show very close performance.

Especially interesting is the classification of speech. For seven features and the linear model the speech classification error is 0.01 and the false alarm rate is 0.07. With 5 s windows no speech samples are misclassified and false alarm rate is 0.03. This can be interesting especially for hearing aids where speech is an important class to recognize.
If longer windows are used better classification can be obtained. Figure 7 shows the relation between window size and test classification error.

![Figure 7](image1.png)

**Fig. 6.** Classification error for both training and test data with 1 s windows. Only a small improvement is achieved with the larger models. A test classification error of just below 0.05 is achieved.

![Figure 7](image2.png)

**Fig. 7.** Test classification error with different window sizes. The minimum classification error is chosen for each model and window size.

5. DISCUSSION

In this paper, a new way of using the pitch for classification is presented. It shows that the pitch can discriminate between the three classes; music, noise and speech. Results of 0.05 general miss rate and 0.01 miss rate of speech were obtained with a 1 s windows. Hence, we find classification results that are significantly better than in earlier systems [1].

The HPS pitch estimator and the linear model are both very efficient algorithms. It might be feasible to incorporate some of the functionality in hearing aids, and this could possibly increase the classification functionality.

6. ACKNOWLEDGEMENT

The authors would like to thank Dan Ellis for making available sound samples on his homepage [8] and KED Timit, CMU_FAF [10] and Keele for clean speech databases.

7. REFERENCES

6.2 On the relevance of spectral features for instrument classification

ON THE RELEVANCE OF SPECTRAL FEATURES FOR INSTRUMENT CLASSIFICATION

Andreas B. Nielsen, Sigurdur Sigurdsson, Lars K. Hansen, and Jerónimo Arenas-García∗

The Technical University of Denmark
DK-2800, Kgs. Lyngby, Denmark
{abn,siggi,lkh,jag}@imm.dtu.dk

1. INTRODUCTION

In the last years there has been an increasing interest in methods that aid music organization and music information retrieval systems, mainly motivated by the large digitalization of music. For a summary of relevant advances in this exciting field, the reader is referred to the website of the series of Music Information Retrieval Conferences.

In this paper, we will pay attention to the problem of instrument classification from the rough audio data (see, for instance, [4]). Among the features that are normally used for this task, those related to the spectral characteristics of the instrument are particularly relevant. We can think of two different models of how the spectrum of a particular instrument changes for different pitches. The first model accepts that the envelope of the spectrum remains constant for all notes, while the second, proposed in [8], states that it is the relation among the amplitude of the harmonics which remains constant. These two models are associated to two set of features: the Mel Frequency Cepstrum Coefficients (MFCCs) and the Harmonic Representation (HR) features.

The two models above are conflicting ones, and, therefore, the main goal of this paper is to illustrate which is the one that better explains the structure of musical instruments. In order to do so, we will train different classification models using both MFCCs and HR features extracted from a rather large database of real instruments recordings [5].

The result of our analysis shows that the models built upon MFCCs outperform those relying on HR. Therefore, MFCCs should be preferred for instrument modelling/classification.

2. SPECTRAL CHARACTERIZATION OF MUSICAL INSTRUMENTS

The spectral structure of a harmonic signal can roughly be divided in two components, as illustrated in Fig. 1: the pitch and the envelope. The pitch is what is perceived as the tone, and its value is given by the fundamental frequency, i.e., the frequency of the first harmonic. The envelope is a modulation of the pitch. If two instruments are playing the same note the pitch will be the same. Under this simplified model it will therefore only be the envelope that makes the two sounds different. Obviously, the pitch changes for different notes, but how the envelope changes is a bit more subtle. Two models are suggested, one that assumes the envelope to be constant, and a second that accepts that it is the relative amplitude of the harmonics that remains constant.

2.1. Constant envelope model: MFCC features

According to this model, the envelope for the spectrum of a particular instrument does not change with the pitch. Therefore, when the pitch is changed the amplitude of each harmonic in the sound varies (see Fig. 2). This model is well...
motivated for some instruments, such as string instruments, by assuming that the pitch is induced by the vibration of the string and the envelope is controlled by the casing, which is of course constant. For other instruments, like trumpets, the validity of the model is not that clear.

It is hard to directly extract the shape of the envelope, but MFCCs capture much of the same information. MFCCs were initially developed for speech, but they are also heavily used in other sound applications, see, for example, [6]. To compute the MFCCs the amplitude of the spectrogram is first found using the Discrete Fourier Transform (DFT) on a small window of the audio data. The modulus of the DFT is then filtered with a Mel filter bank and the logarithm of the outputs is taken. In this way, we obtain a series of numbers related to the energy of the input signal in different frequency bands, whose central frequencies approximate the Mel scale\(^2\). Finally, the Discrete Cosine Transform (DCT) is taken, and the result is the MFCCs.

Since MFCCs consist, roughly speaking, of a DCT of a mel-scaled version of the power spectrum, they contain information about the shape of the envelope of the spectrum. Then, if the envelope were constant, the MFCCs extracted from different windows of the same instrument should be similar, even if they correspond to different notes.

From our explanation, it can be seen that the first MFCC is closely related to the amplitude of the original signal. Therefore, in this paper we will leave out that coefficient, using the values of the next 10 MFCCs to construct the models.

2.2. Constant harmonics amplitude: HR features

This model was suggested in [8], and works under the assumption that it is the amplitude of the different harmonics which remains constant. This means that when the pitch is increased (decreased) the envelope of the spectrum is stretched (compressed) and, therefore, its shape changes, see Fig. 3.

If this model is valid, a good representation for instrument modelling consists simply of the estimated amplitudes of the harmonics, to which we refer in the sequel as Harmonics Representation (HR) features. As we did for the MFCCs, to remove the dependence with the amplitude of the sound signal (i.e., its volume), it is advisable to normalize the amplitude of all harmonics with that of the first one.

The amplitude of each harmonic is directly measurable if the pitch is known. A pitch detector from [7] is used and, together with the labels of the data set and visual inspection of discrepancies, very reliable estimates were produced. The amplitudes of the first 50 harmonics are found, what gives a total of 49 relative HR features.

3. CLASSIFICATION MODELS

In order to study the accurateness of the previous models, we will build multi-class classification models that predict, from both MFCCs and HR features, which instrument is being played. We will use two different classification technologies in order to make our conclusions as general as possible, and to validate that similar conclusions are extracted when using both approaches.

The formulation of the problem can be stated as follows: given a set of $N$ training pairs $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, where $x^{(i)}$ is a vector containing the features extracted from a window of audio data (either MFCCs or HR) and $y^{(i)}$ is a vector of targets containing an ‘1’ in the position associated to the right instrument and zeros elsewhere, the task is to build a function that is able to predict the right targets of new data as accurately as possible.

It is important to remark that the data in our training data sets are strongly unevenly distributed among classes (the number of data in the most numerous class is more than 20 times larger than for the smallest one), thus our classification models should be able to compensate this effect and assume equal priors for all instruments.

3.1. Probabilistic Network

Our first classifier is a multi layer perceptron (MLP) [2] with a single layer of $M$ hidden units and $C$ outputs, each one corresponding to one instrument. The hyperbolic tangent function is used for activation in the hidden units and the softmax function is used in the output units. This fact, together with the use of the logarithmic cost function, makes the network estimate the \textit{a posteriori} probabilities of class membership [8].

To compensate for unbalanced classes we use the following modified cost function:

$$E = -\sum_{i=1}^{N} \sum_{k=1}^{C} \lambda_k y_k^{(i)} \ln \hat{y}_k^{(i)},$$

\(^2\)The Mel scale is related to the perceptual capabilities of the human auditory system.
where $y_k^{(i)}$ is the $k$-th component of $y^{(i)}$, $\hat{y}_k^{(i)}$ is the $k$-th output of the network, and $\lambda_k = 1/N_k$, $N_k$ being the number of samples in class $k$.

The minimization of (1) is carried out using an implementation of the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method.

3.2. Kernel Orthonormalized Partial Least Squares

As a second method, we will consider a kernel based method for multi-class classification. The method consists of two different steps: first, relevant features are extracted from the input data using the Kernel Orthonormalized Partial Least Squares (KOPLS) algorithm [1]; then, a linear classifier is trained to obtain the final predictions of the network.

KOPLS is a method for kernel multivariate analysis that basically works by projecting the input data into a Reproducing Kernel Hilbert Space, where standard OPLS analysis is carried out. To present the method, let us first introduce matrices $\Phi = \{\phi(x^{(1)}), \ldots, \phi(x^{(N)})\}^T$ and $Y = \{y^{(1)}, \ldots, y^{(N)}\}^T$, where $\phi(\cdot)$ is the function that projects input data to some feature space $\mathcal{F}$. Let us also denote by $\Phi = \Phi U$ a matrix containing $n_p$ projections of the original input data, $U$ being a projection matrix of size $\dim(\mathcal{F}) \times n_p$. Then, the KOPLS problem can be formulated as follows (see [1]):

\[
\begin{align*}
\text{maximize:} & \quad \text{Tr}(U^T \Phi^T Y Y^T \Phi U) \\
\text{subject to:} & \quad U^T \Phi^T \Phi U = I
\end{align*}
\]

where the maximization is carried out with respect to $U$.

The Representer Theorem states that $U$ can be expressed as a linear combination of the training data, i.e., $U = \Phi \hat{A}$, and carry out the maximization with respect to $\hat{A}$ instead. However, some advantages in terms of computation and regularization are obtained if we impose a sparse representation for the projection vectors, i.e., we admit that $U = \Phi \hat{U}$, where $\hat{U}$ is a subset of the training data containing only $R$ instances, and $B$ is the new projection matrix of size $R \times n_p$.

Then, the maximization problem for this KOPLS with reduced complexity (rKOPLS) can be stated as:

\[
\begin{align*}
\text{maximize:} & \quad \text{Tr}(B^T K_B YY^T K_B^T B) \\
\text{subject to:} & \quad B^T K_B K_B^T B = I
\end{align*}
\]

where $K_B = \Phi \hat{U} \Phi^T$ involves only inner products in $\mathcal{F}$.

In order to compensate for unbalanced classes, only two modifications to the standard rKOPLS algorithm are needed:

1. All classes should be equally represented in $\Phi_R$.
2. The correlation matrices in (3) should be replaced by their weighted counterparts where all classes have the same influence, i.e.,

\[
\begin{align*}
K_B Y & \quad \leftarrow \sum_{i=1}^{N} \sum_{k=1}^{C} \lambda_k y_{ik} \phi(x_{ik})^T \\
K_B K_B^T & \quad \leftarrow \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_k \phi(x_{ik}) \phi(x_{ij})^T,
\end{align*}
\]

where we have defined $\phi(x_{ik}) = \Phi_R \phi(x_{ik})$.

Once the non-linear features have been extracted from the training data, a single layer perceptron (SLP) with $C$ outputs and softmax activation is trained to learn the relation between these features and the target data, also by minimizing (1) using the BFGS algorithm.

4. EXPERIMENTS

4.1. Data set description and settings

For our experiments we have used a comprehensive database of real instrument recordings, which is available for research purposes at [5]. There are a total of 20 instruments in the data set, all of them recorded at 44.1 kHz and 16 bit/sample. A single note is played at a time, and notes from the complete range of each instrument are included. Moreover, three different amplitude levels are played (pianissimo, mezzoforte and fortissimo). For string instruments there are both arco and pizzicato, and the notes are also played on the different strings. For some of the wind instruments vibrato is also included. We have not included in our data set the pianissimo amplitude level because of the low SNR. Also the pizzicato of string instruments is excluded due to an extremely short duration of the notes. In order for our experiments to be independent from pitch as possible, instruments were requested to share at least one octave. Three instruments were too far away and had to be discarded, leaving 17 instruments for the classification.

The recordings were processed to remove silence periods between notes, and MFCCs and HR were extracted using a window size of 50 ms, which is the time frame on which we do the classifications. This process resulted in a total of 282,812 patterns for training and testing the models. Two different partitions were done for the two sets of experiments described in the next subsections.

Regarding classifier settings, cross-validation was carried out to select the free parameters. For the probabilistic MLP networks (MLP in the sequel) the number of hidden units was set to 30, for which the validation curves were already flat. We found no problems of overfitting, probably because of the size of the data set being used. For the rKOPLS + SLP network (simply rKOPLS in the following), the number of points from each class that are included in $\Phi_R$ was set to 30, also according to the behavior of validation curves. Finally, we used a Gaussian kernel, whose bandwidth was also selected by cross-validation.

As we did for the training of the networks, the accuracy rates that we report in the next subsections are balanced so that all instruments have the same influence on them. Results are averaged over 10 runs of the algorithms.

4.2. Generalization capabilities of the models

In the first experiment, the training data consists of MFCCs/HR extracted from notes spanning the common octave: from B3 to Bb4; all other data is placed in the test data set. Note that the two models of Section 2 tend to agree if the pitch is only slightly modified, while their disagreement is more important for large variations. In this sense, this experiment, where both models are trained using a small range of notes (where they should roughly agree) and tested far away, is a good setting to test their validity.
Table 1. Accuracy rates achieved when training the models using the octave B3-Bb3, and testing outside.

<table>
<thead>
<tr>
<th></th>
<th>MFCCs</th>
<th>HR</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>Tr/Val</td>
<td>91.4 / 79.4</td>
</tr>
<tr>
<td></td>
<td>Tr/Tst</td>
<td>91.2 / 42.8</td>
</tr>
<tr>
<td>rKOPLS</td>
<td>Tr/Val</td>
<td>89.5 / 80.1</td>
</tr>
<tr>
<td></td>
<td>Tr/Tst</td>
<td>85.4 / 34.2</td>
</tr>
</tbody>
</table>

Table 2. Accuracy rates achieved when the training and test data sets are formed with alternating notes.

<table>
<thead>
<tr>
<th></th>
<th>MFCCs</th>
<th>HR</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>Tr/Val</td>
<td>87.4 / 70.7</td>
</tr>
<tr>
<td></td>
<td>Tr/Tst</td>
<td>86.1 / 74.7</td>
</tr>
<tr>
<td>rKOPLS</td>
<td>Tr/Val</td>
<td>89.4 / 78.2</td>
</tr>
<tr>
<td></td>
<td>Tr/Tst</td>
<td>81.4 / 75.9</td>
</tr>
</tbody>
</table>

Fig. 4. Confusion matrices achieved by rKOPLS for the test set of experiment 1. MFCCs on the left and HR on the right.

Cross-validation (CV) in this setting was carried out by using 11 folds, each one consisting of one note of the training data set. Accuracy error rates are reported in Table 1, both for the 11-fold CV (‘Tr/Val’ rows) and for the final training and test error rates (‘Tr/Tst’).

We can first see that 11-fold validation accuracies are much higher than those achieved in the test data set. The fact that both the classifiers based on MFCCs and HR degrade significantly outside the training octave, indicates that both models fail when moving very far away from the training interval. Note however, that not only MFCCs based classifier always get better accuracy rates, but also their degradation with respect to validation rates is much lower (about 50% in comparison to 25% or even less for the classifiers working on HR). The best performance of MFCCs is also clear when looking at the test confusion matrices that are obtained when using the two sets of features (Fig. 4). Therefore, we can conclude that the constant envelope model is a useful approximation to the real behavior of the spectrum of musical instruments, and that MFCCs should be preferred to HR for instrument modelling.

Finally, it is also worth pointing out the consensus between the performance trends shown by MLP and rKOPLS networks, showing that our conclusions are indeed due to the spectral features that are used to feed the classifiers.

4.3. Complete pitch range training

For this experiment the training and test span the whole pitch range of each instrument, with every second note in each set. In this way, we will be able to study the recognition rates that can be achieved from both MFCCs and HRs, if the classifiers are provided with information covering a pitch range as wide as possible. In this case, the training set is divided into 5 folds for validation purposes, each fold taking one out of each 5 notes.

Results for this experiment are displayed in Table 2. Compared to the results of the previous setup, test recognition rates are significantly better, specially when the MFCCs are used, achieving 75.9% recognition rate in combination with the rKOPLS classifier, whose performance is slightly better than that of the MLP network. In relation to previous published studies (see, for instance, [4]) the results in Table 2 look quite competitive, although a direct comparison is not possible given the differences in the nature of the data sets and the experimental settings.

In the light of these results one can conclude that MFCCs are preferable to HR features not only for instrument modelling, but also for automatic classification systems. It also seems clear that, to obtain a classifier of high performance, the training data should include data spanning a pitch range as wide as possible.

5. CONCLUSION

In this paper we have analyzed the spectral structure of musical instruments. Two different models about the behavior of the spectrum of instruments when playing different notes and their associated feature representations, MFCCs and HR, are revisited. Experiments on a rather large data base of real instruments have shown that MFCCs should be preferred to HR, both for musical instrument modelling and for automatic instrument classification.

6. REFERENCES

6.3 Vocal segment classification in popular music

VOCAL SEGMENT CLASSIFICATION IN POPULAR MUSIC

Ling Feng, Andreas Brinch Nielsen, Lars Kai Hansen
Technical University of Denmark
Department of Informatics and Mathematical Modelling
{l, abn, lkh}@imm.dtu.dk

ABSTRACT

This paper explores the vocal and non-vocal music classification problem within popular songs. A newly built labeled database covering 147 popular songs is announced. It is designed for classifying signals from database covering 147 popular songs is announced. It is designed for classifying signals from 1sec-time windows. Features are selected for this particular task, in order to capture both the temporal correlations and the dependencies among the feature dimensions. We systematically study the performance of a set of classifiers, including linear regression, generalized linear model, Gaussian mixture model, reduced kernel orthonormalized partial least squares and K-means on cross-validated training and test setup. The database is divided in two different ways: with/without artist overlap between training and test sets, so as to study the so called ‘artist effect’. The performance and results are analyzed in depth: from error rates to sample-to-sample error correlation. A voting scheme is proposed to enhance the performance under certain conditions.

1 INTRODUCTION

The wide availability of digital music has increased the interest in music information retrieval, and in particular in features of music and of music meta-data, that could be used for better indexing and search. High-level musical features aimed at better indexing comprise, e.g., music instrument detection and separation [13], automatic transcription of music [8], melody detection [2], musical genre classification [10], sound source separation [18], singer recognition [16], and vocal detection [4]. While the latter obviously is of interest for music indexing, it has shown to be an surprisingly hard problem. In this paper we will pursue two objectives in relation to vocal/non-vocal music classification. We will investigate a multi-classifier system, and we will publish a new labeled database that can hopefully stimulate further research in the area.

While almost all musical genres are represented in digital forms, naturally popular music is most widely distributed, and in this paper we focus solely on popular music. It is not clear that the classification problem can be generalized between genres, but this is a problem we will investigate in later work.

Singing voice segmentation research started less than a decade ago. Berenzweig and Ellis attempted to locate the vocal line from music using a multi-layer perceptron speech model, trained to discriminate 54 phone classes, as the first step for lyric recognition [4]. However, even though singing and speech share certain similarities, the singing process involves the rapid acoustic variation, which makes it statistically different from normal speech. Such differences may lie in the phonetic and timing modification to follow the tune of the background music, and the usage of words or phrases in lyrics and their sequences. Their work was inspired by [15] and [19], where the task was to distinguish speech and music signals within the “music-speech” corpus: 240 15s extracts collected ‘at random’ from the radio. A set of features have been designed specifically for speech/music discrimination, and they are capable of measuring the conceptually distinct properties of both classes.

Lyrics recognition can be one of a variety of uses for vocal segmentation. By matching the word transcriptions, it is applicable to search for different versions of the same song. Moreover, accurate singing detection could be potential for online lyrics display by automatically aligning the singing pieces with the known lyrics available on the Internet. Singer recognition of music recordings has later received more attention, and has become one of the popular research topics within MIR. In early work of singer recognition, techniques were borrowed from speaker recognition. A Gaussian Mixture Model (GMM) was applied based on Mel-frequency Cepstral Coefficients (MFCC) to detect singer identity [20]. As briefly introduced, singing voices are different from the conventional speech in terms of spectrometrical and vocal features have differences w.r.t. spectral distribution. Hence the performance of a singer recognition system has been investigated using the unsegmented music piece, the vocal segments, and the non-vocal ones in [5]. 15% improvement has been achieved by only using the vocal segments, compared to the baseline of the system trained on the unsegmented music signals; and the performance became 23% worse when only non-vocal segments were used. It demonstrated that the vocal segments are the primary source for recognizing singers. Later, work on automatic singer recognition took vocal segmentation as the first step to enhance
6.3 Vocal segment classification in popular music

During the course of searching for appropriate features, researchers have realized that system performance can be improved by combining short-time frame-level features into clip-level features. Feature integration is one of the methods to form a long-time feature, in order to capture the discriminant information and characterize how frame-level features change over longer time periods for a certain task. Often the mean and variance of several short-time features are extracted as the clip-level features [17], using multivariate Gaussian model or a mixture of them. However, both the mean-variance and mean-covariance model fail to capture the temporal correlations. A frequency band approach has been proposed in [9], and the energy of the features was summarized into 4 frequency bands. Even though this method can represent temporal development, it does not model the feature correlations.

The multivariate autoregressive model (MAR) was recently introduced to music genre classification [11], and a detailed comparison of different temporal feature integration methods was reported. MAR being able to capture both the temporal correlations and the dependencies among the feature dimensions, has shown its superiority for representing music. We adapt this model in the feature extraction phase on top of short-time MFCCs. Here, a brief description of MAR will be given, for detail, see [11].

Assume the short-time MFCC at time $t$ is denoted as $x_t$, which is extracted from a short period of stationary signals. The MAR can be stated as,

$$x_t = \sum_{p=1}^{P} A_p x_{t-p} + u_t,$$  \hspace{1cm} (1)

where $u_t$ is the Gaussian noise $N(\mu, \Sigma)$, assumed i.i.d. $A_p$ is the coefficients matrix for order $p$; and if it is defined as a diagonal matrix, dependencies among dimensions will not be considered. $P$ indicates the order of the multivariate autoregressive model, meaning that $x_t$ is predicted from the previous $P$ short-time features. It is worth to mention that the mean of MFCCs $m$ is related to the mean of the noise $\nu$ in the following way (note: $I$ is an identity matrix),

$$m = (I - \sum_{p=1}^{P} A_p)^{-1} \nu.$$

3 CLASSIFICATION FRAMEWORKS

We have examined a number of classifiers: linear regression model (LR), generalized linear model (GLM), Gaussian mixture model (GMM), reduced kernel orthonormalized partial least squares (rKOPLS) and K-means.

As the problem is a binary task, only a single dimension is needed for linear regression, and the labels are coded as
we use the softmax function. Otherwise. The prediction is made based on the sign of the output: we tag the sample as a vocal segment if the output is greater than zero; and as a non-vocal segment otherwise.

Generalized linear model relates a linear function of the inputs, through a link function to the mean of an exponential family function, μ = g(ωˈx^n), where ω is a weight vector of the model and x^n is the n′th feature vector. In our case we use the softmax link function, μ_i = \frac{e^{w_iˈx^n}}{\sum_j e^{w_jˈx^n}}, w is found using iterative reweighted least squares [12].

GMM as one of the Bayesian classifiers, assumes a known probabilistic density distribution for each class. Hence we model data from each class as a group of Gaussian clusters. The parameters are estimated from training sets via the standard Expectation-Maximization (EM) algorithm. For simplicity, we assume the covariance matrices to be diagonal. Note that although features are independent within each mixture component due to the diagonal covariance matrix, the mixture model does not factorize over features. The diagonal covariance constraint posits the axes of the resulting Gaussian clusters parallel to the axes of the feature space. Observations are assigned to the class having the maximum posterior probability.

Any classification problem is solvable by a linear classifier if the data is projected into a high enough dimensional space (possibly infinite). To work in an infinite dimensional space is impossible, and kernel methods solve the problem by using inner products, which can be computed in the original space. Relevant features are found using orthonormalized partial least squares in kernel space. Then a linear classifier is trained and used for prediction. In the reduced form, rKOPLS [3] is able to handle large data sets, by only using a selection of the input samples to compute the relevant features, however all dimensions are used for the linear classifier, so this is not equal to a reduction of the training set.

K-means uses K clusters to model the distribution of each class. The optimization is done by assigning data points to the closest cluster centroid, and then updating the cluster centroid as the mean of the assigned data points. This is done iteratively, and minimizes the overall distances to cluster centroids. Optimization is very dependent on the initial centroids, and training should be repeated a number of times. Prediction is done by assigning a data point to the class of the closest cluster centroid.

4 DATABASE

The database used in the experiments is our recently built in-house database for vocal and non-vocal segments classification purpose. Due to the complexity of music signals and the dramatic variations of music, in the preliminary stage of the research, we focus only on one music genre: the popular music. Even within one music genre, Berenzweig et al. have pointed out the ‘Album Effect’. That is songs from one album tend to have similarities w.r.t. audio production techniques, stylistic themes and instrumentation, etc. [5].

This database contains 147 Pop mp3s: 141 singing songs and 6 pure accompaniment songs. The 6 accompaniment songs are not the accompaniment of any of the other singing songs. The music in total lasts 8h 40min 2sec. All songs are sampled at 44.1 kHz. Two channels are averaged, and segmentation is based on the mean. Songs are manually segmented into 1sec segments without overlap, and are annotated second-by-second. The labeling is based on the following strategy: if the major part of this 1sec music piece is singing voice, it is tagged as vocal segment; otherwise non-vocal segment. We believe that the long-term acoustic features are more capable of differentiating singing voice, and 1sec seems to be a reasonable choice based on [14]. Furthermore labeling signals at this time scale is not only more accurate, but also less expensive.

Usually the average partition of vocal/non-vocal in Pop music is about 70%/30%. Around 28% of the 1:41 singing songs is non-vocal music in the collection of this database. Forty-seven artists/groups are covered. By artists in Pop music we mean the performers (singers) or bands instead of composers. The distribution of songs among artists is not even, and Figure 1 gives the total number of windows (seconds) each artist contributes.

5 EXPERIMENTS AND RESULTS

We have used a set of features extracted from the music database. First, we extracted the first 6 original MFCCs over a 20ms frame hopped every 10ms. The 0th MFCC representing the log-energy was computed as well. The means were calculated on signals covering 1sec in time. MAR were afterwards computed on top of the first 6 MFCCs with P = 3, and we ended up with a 6-by-18 A_p matrix, a 1-by-6
vector $\mathbf{v}$ and a 6-by-6 covariance matrix $\Sigma$. Since $\Sigma$ is symmetric, repetitions were discarded. $\mathbf{A}_p$, $\mathbf{v}$ and $\Sigma$ all together form a 135-dimensional feature set. The choice for 6 MFCC is on one hand empirical, and on the other hand to reduce the computational complexity. All in all, for 1 sec music signal we concatenated 135-d MAR, the means of both $0^{th}$ and 6 original MFCCs to form a 142-d feature vector.

5.1 Data Dependency and Song Variation

We used one type of cross-validation, namely holdout validation, to evaluate the performance of the classification framework. To represent the breadth of available signals in the database, we kept 117 songs with the 6 accompaniment songs to train the models, and the remaining 30 to test. We randomly split the database 100 times and evaluated each classifier based on the aggregate average. In this way we eliminated the data set dependencies, due to the possible similarities between certain songs. The random splitting regarded a song as one unit, therefore there was no overlap song-wise in the training and test set. On the other hand artist overlap did exist. The models were trained and test set errors were calculated for each split. The GLM model from the Netlab toolbox was used with softmax activation function on outputs, and the model was trained using iterative reweighted least squares. As to GMM, we used the generalizable gaussian mixture model introduced in [7], where the mean and variance of GMM are updated with separate subsets of data. Music components have earlier been considered as ‘noise’ and modeled by a simpler model [16], thus we employed a more flexible model for the vocal than non-vocal parts: 8 mixtures for the vocal model, and 4 for the non-vocal model. For rKOPLS, we randomly chose 1000 windows from the training set to calculate the feature projections. The average error rates of the five classification algorithms are summarized in the left column of Table 1.

A bit surprisingly the performance is significantly better for the linear models. We show the performance of the chosen classifiers as a function of splits in Figure 2. Each curve represents one classifier, and the trial-by-trial difference is quite striking. It proved our assumption that the classification performance depends heavily on the data sets, and the misclassification varies between 13.8% and 23.9% for the best model (GLM). We envision that there is significant variation in the data set, and the characteristics of some songs may be distinguishing to the others. To test the hypothesis, we studied the performance on individual songs. Figure 3 presents the average classification errors of each song predicted by the best model: GLM, and the inter-song variation is obviously revealed: for some songs it is easy to distinguish the voice and music segments; and some songs are hard to classify.

5.2 Correlation Between Classifiers and Voting

While observing the classification variation among data splits in Figure 2, we also noticed that even though classification performance is different from classifier to classifier, the tendency of these five curves does share some similarity. Here we first carefully studied the pair-to-pair performance correlation between the classification algorithms. In Table 2 the degree of matching is reported: 1 refers to perfect match; 0 to no match. It seems that the two linear classifiers have a very high degree of matching, which means that little will be gained by combining these two.

The simplest way of combining classification results is
by majority voting, meaning that the class with the most votes is chosen as the output. The voting has been done crossing all five classifiers, unfortunately the average voting results (error rates) on the test sets was 18.62%, which is slightly worse than the best individual classifier. The reason seems to be that even though the other classifiers are not so correlated with the linear ones, the miss classification rate is too high to improve performance.

However voting does help enhance the performance, if it performs among not so correlated classification results. Figure 4 demonstrates the sample-to-sample majority voting among GMM, rKOPLS and K-means. The light dash line shows the baseline of random guessing for each data split.

Table 2. A matrix of the degree of matching.

<table>
<thead>
<tr>
<th></th>
<th>LR</th>
<th>GLM</th>
<th>GMM</th>
<th>rKOPLS</th>
<th>K-means</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td>1.000</td>
<td>0.960</td>
<td>0.820</td>
<td>0.804</td>
<td>0.811</td>
</tr>
<tr>
<td>GLM</td>
<td>0.960</td>
<td>1.000</td>
<td>0.814</td>
<td>0.826</td>
<td>0.809</td>
</tr>
<tr>
<td>GMM</td>
<td>0.820</td>
<td>0.814</td>
<td>1.000</td>
<td>0.730</td>
<td>0.774</td>
</tr>
<tr>
<td>rKOPLS</td>
<td>0.804</td>
<td>0.826</td>
<td>0.730</td>
<td>1.000</td>
<td>0.756</td>
</tr>
<tr>
<td>K-means</td>
<td>0.811</td>
<td>0.809</td>
<td>0.774</td>
<td>0.756</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Figure 4. Voting results. It gives the voting performance among GMM, rKOPLS and K-means. The light dash line shows the baseline of random guessing for each data split.

Moreover, we noticed that it is very likely for classifiers to make wrong predictions in the transition sections, meaning the changing from vocal to non-vocal parts, and vice versa. We found this is reasonable comparing with manual labels by different persons, shown in Figure 6. The song was labeled carefully by both people, the absence of mind or guessing should not be a concern. The mismatch indicates the perception or judging difference, and it only happens in the transition parts. The total mismatch is about 3% for this particular song: ‘Bird Guhl’ by Antony and the Johnsons.

5.3 ‘Artist Effect’

In previous experiments, we randomly selected songs to form training and test sets, hence the same artist may appear in both sets. Taking the previous results as a baseline, we studied the ‘artist effect’ in this classification problem. We tried to keep the size of test sets the same as before, and carefully selected around 30 songs in order to avoid artist overlap for each split, and formed 100 splits. The second column of Table 1 summarizes the average error rates for 4 clas-
6.3 Vocal segment classification in popular music

ISMIR 2008 – Session 1c – Timbre

fiers. The average results are a little worse than the previous ones, and they also have bigger variance along the splits. Therefore we speculate that artists do have some influence in vocal/non-vocal music classification, and the influence may be caused by different styles, and models trained on particular styles are hard to be generalized to other styles.

6 CONCLUSION AND DISCUSSION

We have investigated the vocal/non-vocal popular music classification. Experiments were carried out on our database, containing 147 popular songs. To be in line with the label set, the classifiers were trained based on features at 1sec time scale. We have employed 142-d acoustic features, consisting MFCCs and MAR, to measure the distinct properties of vocal and non-vocal music. Five classifiers have been invoked: LR, GLM, GMM, rKOPLS and K-means.

We cross-validated the entire database, and measured the aggregate average to eliminate the data set dependency. GLM outperformed all the others, and provided us with 18.46% error rate on the baseline of 28%. The performance has great variation among data splits and songs, indicating the variability of popular songs. The correlations among classifiers have been investigated, and the proposed voting scheme did help among less correlated classifiers. Finally we looked into the ‘artist effect’, and it did degrade the classification accuracy a bit by separating artists in training and test sets. All in all vocal/non-vocal music classification was found to be a difficult problem, and it depends heavily on the music itself. Maybe classification within similar song styles can improve the performance.

7 REFERENCES


Chapter 7

Structure Learning

7.1 Synchronization and comparison of Lifelog audio recordings

SYNCHRONIZATION AND COMPARISON OF LIFELOG AUDIO RECORDINGS
Andreas Brinch Nielsen, Lars Kai Hansen
Technical University of Denmark
DK-2800, Kgs. Lyngby, Denmark
{abn,lkh}@imm.dtu.dk

ABSTRACT
We investigate concurrent ‘LifeLog’ audio recordings to locate segments from the same environment. We compare two techniques earlier proposed for pattern recognition in extended audio recordings, namely cross-correlation and a fingerprinting technique. If successful, such alignment can be used as a preprocessing step to select and synchronize recordings before further processing. The two methods perform similarly in classification, but fingerprinting scales better with the number of recordings, while cross-correlation can offer sample resolution synchronization. We propose and investigate the benefits of combining the two. In particular we show that the combination allows sample resolution synchronization and scalability.

1. INTRODUCTION
LifeLogs are extended digital recordings of a persons life. This could for example include (e)mail correspondence, visited web sites, documents, chat logs, and video and audio recordings. The typical and original aim [1] of such recording is to boost recollection of events. Modern examples include MyLifeBits [2], LifeStreams [3] and Lifelogs [4]. Extensive digital recording could also be used for modelling behaviour as in, e.g., [5].

The task of collecting and processing LifeLog data stores is huge, and here we focus on audio aspects, as pioneered by Ellis and coworkers, see e.g., [6]. While conventional LifeLogs concern organization of personal archives we are particularly interested in the group perspective, and thus expand the scenario from including only the recordings of a single individual to integrate the recordings of multiple subjects. Multi-subjects audio analysis has been pursued earlier, e.g., in the context of conversational patterns as in [7]. We envision a setup in which employees wear microphones recording continuously while at work. Because we imagine microphones worn by individuals we can not only say something about who said what, but also estimate who actually received given information, i.e., who heard what? Here we will not be concerned with the obvious ethical issues involved in storing such audio but only investigate the mounting technical challenges.

Signal processing of multi-microphone recordings has a significant literature, see e.g., work on signal separation [8], and also includes work on distributed microphone arrays from specially equipped rooms [9]. Common to most of these is that the recordings are well synchronized and that they are recorded within the given locality thus in principle contains the same acoustic environment, in addition it is often assumed that the actual microphone placement is fixed and known. In our setting of ‘moving microphones’ some of the parameters must be inferred from the data itself. In this paper we will consider two aspects of concurrent Lifelogs, namely 1) to classify recordings as being from within the same area, meaning, that they have recorded the same audio events, and 2) we will investigate synchronization of recordings. Synchronization is necessary, because of the distributed nature of the recordings and wanted for subsequent blind signal separation processing. Recording devices that are not linked are likely to produce timing differences of the order of seconds, which will make un-mixing filters invoked by ‘convolutive’ blind signal separation algorithms prohibitively long.

The paper is organized as follows, in section 2 we describe two different audio similarity measures, cross-correlation and fingerprinting. In section 3 the classification problem is described. Different approaches are investigated, including one-on-one classification, a joint approach using both similarity measures and a joint classification scheme that assures a block diagonal mixing matrix. In section 4 experiments are performed within a large public data set from the AMI corpus [9] and own real-room experiments.

2. AUDIO SIMILARITY MEASURES
In this section the two measures of similarity will be presented. The normalized cross-correlation coefficient is a well known statistical quantity. The fingerprinting procedure is less so and was originally presented to identify pop songs recorded with a cell phone and compared to a large database.
7.1 Synchronization and comparison of Lifelog audio recordings

The two methods return a measure of the similarity between two signals. A binary decision is necessary determining whether the two signals have been recorded from the same environment. This will be achieved through the training of a classification algorithm and will be presented in the next section.

Cross-correlation. The sampled cross-correlation function is given as,

\[ xc(m) = \frac{1}{N\sigma_1\sigma_2} \sum_n x_1(n)x_2(n - m). \]

Where the cross-correlation is normalized with the product of the standard deviations \(\sigma_1, \sigma_2\) such that the range is \([-1; 1]\).

As the signals have different delays and possibly have been filtered differently through different sound paths, a negative cross-correlation is as significant as a positive and thus the absolute magnitude is used. This produces a measure in the range \([0; 1]\), where the value 1 is for similar signals which surely come from the same environment while the value 0 is no correlation at all, meaning that the recordings are likely to come from different audio environments.

The cross-correlation is computed as a function of lags in the range \([-10s; 10s]\). The cross-correlation coefficients of two recordings from the same room as a function of the lag is shown in fig. 1. The lag location and value of the maximum are found. The maximum value will be used to make the binary decision, and the lag can be used to minimize the delay between recordings.

Fingerprinting. This method was proposed in [10] and used in [11]. It was originally intended for recognizing songs from short cell phone musical recordings. The fingerprint method preprocesses a recording in a way to drastically reduce the dimensionality and ease the accessibility for future comparisons.

Fig. 1. The comparison of two recordings of the same meeting as a function of the lag using the normalized cross-correlation. The SNR is very good in this example, although multiple peaks exist showed in the zoomed view (insert). Only the value and location of the maximum is used as indicated.

For each recording a number of hashes are generated, together with their associated time stamps. When two recordings are compared, all hashes are compared and when hits are found, the time difference between their time stamps is saved. A histogram is made of the time differences. If two recordings are from the same environment a relative large number of hits is expected to occur with the same time difference, and this will show up in the histogram as in fig. 3. The histogram is processed as the cross-correlation, i.e., the maximum is found and saved together with the value of the lag. While the cross-correlation function produces a lag with ‘sample resolution’ the resolution in the fingerprinting procedure is depends on histogram and hash settings and in the current setup amounts to approximately 50 ms.

The hashes are generated from landmarks in the spectrogram. Each frequency bin is normalized (over time) to zero mean, which reduce the effect of the in general higher energies in the lower frequency bands. To ensure a more uniform distribution of landmarks over the spectrogram, it
Fig. 3. The same setup as in figure 1 but using the fingerprinting method instead. Obviously the two methods return similar results, but they differ greatly in the calculations. The same clear SNR is present, but because of the lower resolution only a single peak is present in the zoomed view.

is coarsely divided in both time and frequency in a number of equally sized parts. In each of these parts local maxima are found. The $k$ largest local maxima in each part is recorded. We found that a small amount of smoothing, prior to locating maxima improved results, thus a 3x3 moving average filter was applied.

Within the parts of the spectrogram each unique pair of the $k$ landmarks ($k(k - 1)/2$ pairs) is used to generate a hash. Each hash consists of three $b$ bit values; the (absolute) time difference between the two landmarks, the frequency value of the first landmark and the frequency value of the second landmark. All three values are discretized to $b$ bit, and concatenated into a $3b$ bit hash. The time point of the first landmark is saved together with the hash.

3. CLASSIFICATION OF AUDIO SIMILARITY

Previously, measures of similarity between two recordings was described. The next step will be to decide when the measured similarity is significant and the recordings are considered to be from the same environment, which is a classification problem. The measures are one dimensional for both methods. Histograms of the similarities of the training set, and this does not reflect a ’prior’, it is important to normalize the classification error during training. Otherwise skewed results will be obtained. In this case it is straightforward, the number of false negatives and false positives are simply divided by the appropriate sample sizes before being added together to compute the classification error rate.

3.1. A combined approach

The two methods reviewed in the previous section differ in the resolution of the delay and in scalability with increasing number of sources. The experiments will show that the two methods perform comparably in classification and this would point to recommending cross correlation because of the increased resolution. However, when comparing recordings one-on-one, the number of comparisons will always increase quadratically with the number of recordings. This is the situation for both methods, but the complexity of the comparisons differ. The cross correlation has all the complexity in the comparison stage and is therefore severely hurt for many recordings. The fingerprinting method preprocesses the data to make the comparisons relatively light. The preprocessing is heavier than for the cross-correlation, but the preprocessing only scales linearly with the number of recordings and therefore, for increasing number of recordings the fingerprinting method will perform significantly faster. This is illustrated in fig. 5.

To use this timing advantage a combination is proposed, working in two stages. In the first stage the fingerprinting method is used to make a coarse classification. In the second stage the cross correlation is used only on the recordings that were classified as coming from the same environment in the first stage. The results from the cross correlation are used both to check the classification and the increased delay resolution is used to precision synchronize the recordings.
7.1 Synchronization and comparison of Lifelog audio recordings

3.2. Joint classification of multiple recordings

In the previous section, classification was done based on pairwise comparison. This could result in source 1 and 2 being similar, source 2 and 3 being similar, but source 1 and 3 being dissimilar. How do we interpret this result? We need to group the sources consistently and we will not allow overlapping clusters. First the similarity measures are set up in a similarity matrix containing all the two by two comparisons. This could result in source 1 and 2 being similar, source 2 and 3 being similar, but source 1 and 3 being dissimilar. How do we interpret this result? We need to group the sources consistently and we will not allow overlapping clusters.

A block diagonal form will be obtained by a greedy procedure similar to Ward’s agglomerative clustering [12]. Because the similarity matrix is symmetric only the upper triangle of the matrix is considered. First, we locate the maximum similarity and the two involved recordings are grouped together. This cluster will not be split again and therefore the mean of similarities of the two recordings to the remaining recordings are calculated and entered in the similarity matrix. Then the next maximum is found, and connected either to the existing group or to another single source thereby creating a new cluster. This procedure is continued until a threshold is reached. This threshold will be trained using line search, and the same normalized classification error rate measure from before. A simple example is shown in table 1.

**Table 1.** Block diagonal classification. 8 is the maximum and recording three and four are clustered together, and the mean of the similarities to other recordings are calculated. Next, 7 is the maximum and recording one and two are clustered. For a threshold larger than four the clustering ends here, otherwise a final step will join the two clusters.

The AMI corpus [9] is a large collection of multimodal meeting recordings including multiple audio recordings with different microphone configurations including microphones attached to individuals (lapel microphones). Two meetings have been used for training and two for testing. Each meeting has four participants and the recordings are from the lapel microphone, so four channels are available from each meeting. Two meetings are used together to make eight channels from two different settings.

The data is framed in 30 second frames, which are overlapping by 20 seconds. The training set was a little more than two hours long, giving 786 frames. The test set was 87 minutes long giving 524 frames. The recordings are downsampled to 8 kHz, and the range of possible delays is set to 10s.

For the fingerprinting method the spectrogram is divided into smaller parts as explained previously. The frequency axis is divided in two and the time axis is divided into 1 s long windows (30 windows). In each part \( k = 10 \) landmarks are found, and a total of \( k(k-1)/2 \cdot 30 = 2700 \) landmarks are generated per frame. The spectrogram is computed with 256 samples and 195 samples overlap, so that each part used for the landmarks becomes \( 64 \times 127 \), and the three parts of the hash are discretized to \( b = 6 \) bit each.

To simulate less ideal situations, e.g., poor microphones or recording devices, uncorrelated gaussian noise was added in different signal-to-noise ratios (SNR). The SNR is calculated within each 30 s window.

![Fig. 5. The time consumption of the different proposed algorithms. Time ratio means the time it takes to process one unit of time - 1 means that processing time equals the duration of the recordings. A quadratic increase can be observed for cross correlation and a linear increase for the fingerprinting method. The proposed combination results in time consumption in between the two.](image-url)
Training was done using line search and grid search for the combined approach. In fig. 6 an example of such training can be seen. It can be seen that besides the computational advantage, an additional small reduction of the training error can be gained, since the minimum exploits the combined model space. The cross correlation threshold is driven closer to zero, whereas the fingerprint threshold only moves slightly, compared to the case when only one of the measures is used.

4.1. Results

Obviously the classification performance is important, but because of the extensive data sets resulting from Lifelogs execution time is also a concern.

Results are shown in fig. 7 (A). For low noise conditions, the fingerprinting algorithm provides the best performance of the two algorithms with a test error rate below 0.01. The method takes a significant hit in performance for additive noise and ends up around 0.05 in test error rate. Cross correlation works significantly worse with a test error rate close to 0.02 which is around twice as much as the fingerprint method. The algorithm is however quite insensitive to noise due to the robustness against addition of uncorrelated gaussian noise.

If the method is used as a preprocessing step before ICA or similar evaluations, we are interested in clustering the recordings, and a block diagonal classification is relevant. The results in this setting is shown in figure 7 (B). Similar trends are found, but the fingerprinting is lot less sensitive to noise in this case. In both figures the combined approach is plotted as well. For training we expect that it performs better than the other two basic methods. We see that the performance also translates to an improved test error. This means that the good performance in low noise conditions of the fingerprinting procedure and the good performance of cross correlation in relatively high noise settings both can be achieved in the combined approach.

In figure 5 the time complexities of the algorithms are shown. The cross correlation scheme shows a quadratic growth in time, but the fingerprinting method shows close to linear growth. The reason for this is, that most of the processing time of the fingerprinting method lies in the computation of the fingerprints, whereas the actual comparisons are very fast because of the hashing structure and the severely reduced data size. In the present case, the reduced size is actually the only explanation, since a proper index structure was not used to facilitate the efficient search. The number of fingerprint computations only increases linearly with the number of sources, hence the linear increase in computational time. The combined approach uses this fact, since the fingerprinting is done full the combined approach uses more time than fingerprinting alone, but is significantly faster than cross correlation.

4.2. Real experiment

As a final experiment, recordings were done of routine office work. Two PDAs were used one placed on the lapel of a student and another placed in the office. About 90 minutes of recordings were done. Some of the elapsed time was spend working in the office (same environment), and the rest of the time consisted of a meeting outside of the office (different environments). The results are shown in figure 8. The decision thresholds from the training session were used. As is evident in the figure, the system is able to classify whether two recordings are from the same environment. The error rate is 0.017 which is very similar to the found test errors. All errors are in the ‘same environment’ states.

Manual inspection of the found lags was performed as well, but the ground truth is not available, so a performance value is not available. The third plot shows the reported lags and as can be seen they are quite constant indicating that the
Fig. 8. The top figure shows the manually tagged labels, and the second shows the estimated labels. Accurate classification is achieved with an error rate of 0.017. Four errors are found in the beginning of the recording (A) because of handling noise in one recording when attaching the microphone to the lapel. Remaining errors are single points scattered randomly in parts from ‘different environments’ (B). The third plot shows the estimated delays. A peak is seen in the lags (C) which could be a case of a side peak (cf. figure 1). A decreasing trend is observed in the delays indicating that the sample frequencies do not exactly match for the two recording devices, viz. 2.717 ms is lost. The used pda’s had their clocks synchronized just before starting the experiment and still a delay of 0.4 s is found. Such delays would have a severe impact on, e.g., a convolutive ICA algorithm further substantiating the need for synchronization.

A ‘true lag’ is found. An interesting trend can be observed i.e. the lag changes slightly over time. The reason for this is probably that the clocks in the two devices are not accurate and therefore the sampling rates are slightly different between the two recordings causing a drift.

5. CONCLUSION

In this paper the first steps toward the analysis of multiple Lifelog audio recordings were taken. The steps included clustering recordings into joint environments followed by a synchronization step of recordings within a given audio event group. Two approaches were investigated, showing similar classification performance, but having different advantages. Cross-correlation has sample precision in the found delays and thus can give more accurate synchronization. The fingerprinting method scales much better with the number of recordings. The time complexity is likely to be a serious challenge for real world applications. A joint approach of the two methods was implemented and obtained both the accurate sample resolution and an increased execution speed.

Using the method suggested here the subsequent ICA blind separation problem could be limited to two four-source recordings, instead of one eight-source recordings. For convolutive ICA the shorter the convolutive filters are, the faster and better results are typically obtained. By synchronizing the recordings prior to solving the problem, the filters are limited to only capture the inherent delay from the different distances between microphones and sources. This greatly shortens the length of the filter, from potentially 5 − 10 s to below 1 s.

An experiment was performed in a real setting and showed that it is indeed possible to detect audio environment similarities and to synchronize the recordings.

6. REFERENCES

7.2 Structure learning by pruning in independent component analysis

Structure learning by pruning in independent component analysis

Andreas Brinch Nielsen*, Lars Kai Hansen

Informatics and Mathematical Modelling, Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark

Available online 15 February 2008

Abstract

We discuss pruning as a means of structure learning in independent component analysis (ICA). Learning the structure is attractive in both signal processing and in analysis of abstract data, where it can assist model interpretation, generalizability and reduce computation. We derive the relevant saliency expressions and compare with magnitude based pruning and Bayesian sparsification. We show in simulations that pruning is able to identify underlying structures without prior knowledge on the dimensionality of the model. We find, that for ICA, magnitude based pruning is as efficient as saliency based methods and Bayesian methods, for both small and large samples. The Bayesian information criterion (BIC) seems to outperform both AIC and test sets as tools for determining the optimal dimensionality.

Keywords: Independent component analysis; Pruning; Structure; Sparsity

Independent component analysis (ICA) is a topic of significant current interest in many scientific areas, see, e.g., [16]. ICA is used in a variety of signal processing applications such as reconstruction of sound sources from multi-microphone recordings, image processing, as well as for understanding EEG signals from multiple independent sources in the human brain. ICA has found numerous uses in data analysis ranging from text mining to bioinformatics and modelling of music, see, e.g., the recent conference proceedings [25].

Sparsity in various forms has been investigated in the framework of ICA. In [13] a so-called sparse component analysis is used to find the number of sources of a mixing matrix with an over estimated number. In [21] sparseness of the filter vectors in convolutive ICA is sought, and in [14] sparse encoding of handwritten digits are found. In the recent work on ICA based causal modelling [27,15] a sparse Markovian structure is sought, causing the separation matrix to become lower triangular or a row/column permutation of a lower triangular matrix.

In this work we will learn the structure of the mixing matrix possibly leading to sparse solutions. By the structure is meant the number and placement of non-zero parameters. Structure learning is of interest in ICA of abstract data and in causal inference where it can assist model interpretation, see, e.g., [28,10,27,15]. Furthermore, structure learning is relevant to classical blind signal separation applications. In ICA of complex multi-agent visual or auditory scenes, sources can be present only in a subset of the recorded mixtures. For example in video where objects can be obscured [4]. In blind separation of audio mixtures of sources in motion a source can be blocked by a physical barrier so that it practically disappears from the scope of a microphone. In general, structure learning reduces complexity and can reduce overfit in short data sequences, hence improves generalizability by Occam’s principle [3]. ICA complexity control at the level of estimation of the number of sources has earlier been shown to improve generalizability, see, e.g., [10]. Finally, we note that structurally optimized models may reduce the computational burden.

The approach taken here is that of parameter pruning, which leads into two questions. Which parameters to prune and how many? These questions will be dealt with one by one. Pruning typically proceeds by eliminating the least salient parameters, retraining the reduced model, and repeating the procedure until a minimal configuration is obtained. This leads to a nested sequence of increasingly
sparse models. To assess the saliency of the parameters, techniques from other machine learning contexts will be used; magnitude based pruning where the saliency is set equal to the magnitude of the parameter, optimal brain damage (OBD), optimal brain surgeon (OBS) and automatic relevance determination (ARD), see, e.g., [3]. Selecting the optimal model within the derived sequence of pruned models can be done in several ways. Here we will compare test set validation, Bayes information criteria, Akaike’s information criteria and the Laplace approximation of the posterior model likelihood [3].

The paper is organized as follows. First we outline the basic ICA model in Section 1. In Section 2 pruning based on estimated saliency will be reviewed. Evaluations of the different saliency measures is done using synthetical data and assuming known number of zeros in Section 3. This will show if the measures can identify the zero parameters. In Section 4 we will test the model selection criteria, to see if they can identify the correct number of zero parameters, and finally, in Section 5, we present a more elaborate simulation experiment to demonstrate the efficiency of pruning based structure learning in auditory scene analysis.

1. Independent component analysis

The basic independent component model involves a linear mixture of L sources into L observations, X = AS, where S = [s1, . . . , sL] is the source matrix, X = [x1, . . . , xL] the observation matrix, sL and xL the source and observation L-vectors at time n. A is a square L × L mixing matrix. Assuming that the source signals are i.i.d.,

\[ p(s) = \prod_i p(s_i) \]

we obtain the likelihood function

\[ p(X|A) = \prod_i p(x_i|A) = \prod_i \int p(x_i|s_i, A) \prod_i p(s_i) ds_i. \]

Introducing the separation matrix W = A−1 the likelihood can be written as

\[ p(x|s, A) = \delta(x - As), \]

\[ p(x|W) = \int \delta(x - W^{-1}s) \prod_i p(s_i) ds_i = |W| \prod_i P \left( \sum_j W_{ij} s_j \right). \]

| is the absolute determinant. We will assume a simple parameter free source distribution for computational convenience, \( p(s) = 1/\pi \cosh(v) \). This distribution has a long tail (super-Gaussian), and has been seen to work well for ICA of other long tailed distributions as well, including speech and music [2,22]. The negative log-likelihood can be written in terms of either the mixing or the separation matrix

\[ \mathcal{L}(W) = - \log p(X|W) = - N \log |W| + \sum_i \log \cosh \left( \sum_j W_{ij} s_i \right) + NL \log \pi, \]

\[ \mathcal{L}(A) = - \log p(X|A) = N \log |A| + \sum_i \log \cosh \left( \sum_j A^{-1}_{ij} s_i \right) + NL \log \pi. \]

We base estimation of the mixing parameters on maximum likelihood (ML) leading to Infomax-like algorithms [2,5]. As discussed in [22] optimization of the mixing or separation matrices do not necessarily produce equivalent results. In the context of sparsity there is further the issue of whether we seek sparsity of the mixing A or separation W matrices. Here we will discuss both options, and equations for the error functions and their derivatives are given in Appendix A. The notation \( \mathcal{L}(\theta) \) is used where the derivations can be done for either \( \mathcal{L}(A) \) or \( \mathcal{L}(W) \).

2. Sparsity by pruning

Pruning has been widely used for simplification of neural network architectures [18,29,12,6,23,30,24,7,17]. Pruning typically proceeds from some over-parametrized model which is simplified in a sequence of pruning/retraining steps leading to some minimum null configuration. The sequence of nested architectures can then be inspected for performance to obtain a given objective, e.g., cross-validation performance. The pruning process can be seen as an efficient heuristic substituting for a complete search through all possible architecture subsets of the initial configuration.

The first and simplest approach to importance ranking is by parameter magnitude [18] and it has recently been proposed for ICA in [27]. In [18], however, it was shown that for optimization of neural network architectures for generalizability, significantly better results can be obtained using the saliency concept. The key idea is to compute the increase in the cost function—or equivalently the decrease in likelihood—incurred by setting a given parameter to zero. One way to achieve this is by actually setting the parameter to zero and recompute the error, this has been referred to as ‘brute force’ pruning [29]. To reduce the computational burden, ranking based on estimated saliency has been proposed.

We note that if model selection is based on either probability using the Bayesian information criterion or estimated by Akaike’s information criterion, these criteria take the form of the training set log-likelihood plus a penalty term that is only a function of the number of parameters, and is optimized if we remove the parameter...
that incurs the least decrease in the training set log-likelihood. Hence, the saliency should in these cases be based on the likelihood cost.

2.1. Optimal brain damage

In the so-called OBD scheme [18] the saliency estimator is based precisely on an expansion of the cost function that can be the negative log-likelihood

$$E = E_0 + \Delta \theta^T \frac{\partial E}{\partial \theta} + \frac{1}{2} \Delta \theta^T \mathbf{H} \Delta \theta + c(\Delta \theta).$$

$$\mathbf{H} = \nabla^2 E / \partial \theta^2$$ is the Hessian matrix of second derivatives. If exactly one of the weights is set to zero, $\Delta \theta_i = -\theta_i$, the increase in error can be found to be

$$\Delta E_i = -\theta_i^2 \left[ \frac{\partial E}{\partial \theta_i} + \frac{1}{2} \theta_i^2 \mathbf{H}_{ii} \right] + c(-\theta_i^2).$$

If the set of weights used for expansion are obtained by cost minimization, the first term can be assumed zero. If we further neglect third order terms the price of setting a single weight to zero can be estimated by the OBD saliency,

$$\Delta E_i \sim S_i = \frac{1}{2} \theta_i^2 \mathbf{H}_{ii}.$$

2.2. Optimal brain surgeon

This method is an extension of OBD. It is reasonable to assume that some of the increase in error due to the deletion of a weight could be countered by retraining the remaining weights. Hassibi and Stork [12] noted that within the second order approximation it is possible to estimate the result of both deleting a parameter and retraining the remaining parameters. Setting a weight to zero can be written as $\Delta \theta_i + \theta_i = e_i^T \Delta \theta_i + \theta_i = 0$, where $e_i$ is the unit weight vector. Minimizing the error function disregarding higher order terms, assuming zero first order term and under the constraint $e_i^T \Delta \theta_i + \theta_i = 0$ leads to the so-called OBS saliency:

$$S_i = \frac{1}{2} \theta_i^2 \mathbf{H}_{ii}.$$  

2.3. Automatic relevance determination

Sparsification based on parameter priors can be cast in terms of the so-called pruning priors [8,9] which is equivalent to shrinkage regression with an L1 norm regularization, see, e.g., [31]. Alternatively, hierarchical priors with adaptive hyper-parameters can lead to pruning [11]. This mechanism was used in [32] to obtain sparse Bayesian models. The basic mechanism is also referred to as ARD, see, e.g., [19]. We here review the basic results, see also [3] for additional detail.

We consider the joint pdf of data and parameters obtained by introducing a prior on the parameters:

$$p(X, \theta | z) = p(X | \theta) p(\theta | z),$$

where $\theta$ can be either $\mathbf{A}$ or $\mathbf{W}$ and $z$ is the regularization parameter vector. We will use the conventional Gaussian prior with the hyper-parameters playing the role of precisions

$$p(\theta | z) \propto \exp \left( - \frac{1}{2} \sum z_i \theta_i^2 \right).$$

When minimizing the negative logarithm of the joint pdf

$$\mathcal{L}(\theta, z) = - \log p(X | \theta) p(\theta | z),$$

the maximum a posteriori (MAP) estimates are found.

To estimate the value of $z$, the posterior distribution is found:

$$p(z | X) = \int p(X | \theta) p(\theta | z) d\theta.$$

Assuming a uniform prior over $z$, only $p(X | z)$ needs to be maximized, and can be found by marginalizing the likelihood:

$$p(X | z) = \int p(X | \theta) p(\theta | z) d\theta.$$  

This integral cannot be solved directly for the ICA problem, but the Laplace approximation can be used. The Laplace approximation uses a second order Taylor expansion of the logarithm of the joint pdf around the MAP estimate of the parameters, i.e., approximates the joint pdf, $p(X | \theta) p(\theta | z)$, by an unnormalized Gaussian distribution. Because the MAP estimate is a maximum, the first order derivative vanishes:

$$p(X | z) = \int p(X | \theta) p(\theta | z) d\theta$$

$$\approx e^{-\mathcal{L}(\theta_{MAP}, z) / 2} \log |H| \cdot 2^n \log 2 \pi. \quad (1)$$

$$\Delta \theta = \theta_{MAP} - \theta, \quad \mathbf{H} = \nabla^2 \mathcal{L}(\theta, z) / \partial \theta^2 \quad \text{and} \quad \| \text{is the determinant.}$$

$$\Delta \theta = \theta_{MAP} - \theta, \quad \mathbf{H} = \nabla^2 \mathcal{L}(\theta, z) / \partial \theta^2 \quad \text{and} \quad \| \text{is the determinant.}$$

The system is trained with respect to the (hyper) parameters in an EM-like iterative scheme; first, the maximum posterior weights are estimated for fixed $z$, by minimizing of $\mathcal{L}(\theta, z)$. Next, we re-estimate $z$ for fixed $\theta$:

$$\mathcal{L}(- \log p(X | z)) = \mathcal{L}(\theta_{MAP}, z) - \frac{1}{2} \log |\mathbf{H}| - \frac{1}{2} L \log 2 \pi \quad (1)$$

$$\frac{\partial \mathcal{L}(\theta_{MAP}, z)}{\partial \theta_{MAP}} = \frac{1}{2} \left[ \mathbf{H}_{\theta_{MAP}}^2 - 2 \frac{\partial \log |\mathbf{H}|}{\partial \theta_{MAP}} \right] = 0$$

$$\Rightarrow \Delta \theta_{MAP} = \frac{1}{2} \left[ \mathbf{H}_{\theta_{MAP}}^2 - 2 \frac{\partial \log |\mathbf{H}|}{\partial \theta_{MAP}} \right]$$

The derivative of $\log |\mathbf{H}|$ is

$$\frac{\partial \log |\mathbf{H}|}{\partial \theta_{MAP}} = \text{Tr} \left( \mathbf{H}^{-1} \frac{\partial \mathbf{H}}{\partial \theta_{MAP}} \right) = [\mathbf{H}^{-1}]_{\theta_{MAP}}.$$
because $H = V^T_d Z(\theta) + \text{diag}(a)$, Eq. (2) could be 'solved' and the optimal hyper-parameters found in one step; however, practise has shown that this is too greedy. Therefore hyper-parameters and parameters are updated iteratively [3].

For a Gaussian weight prior the individual hyper-parameters can either converge to some finite value, hence producing finite weights or they can escape to infinity, implying a delta-like posterior distribution around zero for the corresponding parameters, i.e., effectively pruning the associated parameter away [11,19]. Expressions for $L(\theta, x)$, $\partial L(\theta, x)/\partial \theta$ and $\partial^2 L(\theta, x)/\partial \theta^2$ are given in Appendix A for both $\theta = A$ and $\theta = W$.

### 3. Quality of ranking

In the previous section we detailed some different approaches for assessing the importance of a parameter. In this section we will evaluate the different methods. We will assume that the true number of non-zero parameters is known and therefore only test which parameters should be pruned, and not how many. How many parameters to prune is the subject of Section 4, and in Section 5 a possible application is simulated. The pruning is performed pruning the known number of parameters. Then a miss classification rate is calculated as the number of correctly identified zero parameters divided by the known number of zeros:

$$\text{miss rate } 1 = \frac{1}{\text{#experiments}} \sum \frac{\text{# missed zeros}}{\text{# true zeros}}$$

We will consider pruning based on magnitude (MB), ARD, OBD and OBS for a synthetic data set. The sources (S) are generated using the $p(s) = \frac{1}{\pi \cosh(s)}$ distribution. A random mixing (A) or separation (W) matrix is generated with a number of zeros and is used to generate the observations, $X = W^{-1}S = AS$. The non-zero entries are generated using $W = r + \frac{1}{2} \text{sign}(r)$, where $r \sim \chi(0, 1)$. In the experiment $L = 5$ sources are chosen. To gauge the dependence on sample size we vary the number of observations $N \in [30; 1000]$. We use the Broyden-Fletcher-Goldfarb-Shanno (BFGS) [20] algorithm to optimize the log-likelihood.

In ARD based pruning the procedure is carried out once (iterating weights and hyper-parameters to convergence) and the resulting hyper-parameters are used to define the ranking order for deletion towards the desired number of non-zero elements. For MB, OBD and OBS we prune one weight at a time until the desired number is achieved, including intermediate re-training steps of each partially pruned model. The experiment is repeated 1000 times with different random sources and mixing/separation matrices. In this study we have created ‘true’ models that are lower triangular. Since a lower triangular matrix has a lower triangular inverse this corresponds to the Markovian structures investigated in [15].

### 3.1. Results

We report in Fig. 1 the miss classification rate as a function of the number of samples in the training data. Both for estimating mixing and separation matrices we find that performance is strongly dependent on the amount of data provided. The maybe most surprising result is that in both cases the magnitude based method outperforms the more involved estimators with OBD being significantly poorer in the range of larger sample size where the performance of the three other methods are more or less at par.

![Fig. 1. Miss classification rate over 1000 pruning sequences. The rate is reported vs. sample size. The ICA problem is square with $D = 5$, the true mixing matrix is lower triangular, hence having 10 zeros. In (A) we show performance for pruning based on the separation matrix while in (B) the result is based on the mixing matrix. Performance is strongly dependent on sample size. While performances of magnitude based, ARD and OBS are comparable, OBD seems to fall short even after learning in relatively large data sets.](image-url)
To understand the properties of the Hessian we can pass to the large sample limit, by replacing sums over \( n \) with expectations over the assumed distribution of the source signals. The Hessian (parameterized using the separation matrix) then becomes

\[
\hat{H}_{i,i} = N \left( \sum_i A_{i,j} A_{i,j}^T + \delta_{i,j} \sum_i A_{i,j} A_{i,j}^T \right) / C_{0}^2 + \delta_{i,j} \sum_i A_{i,j} A_{i,j}^T.
\]

The expectation value is given by

\[
\mathbb{E}(\hat{H}_{i,i}) = N \left( \frac{1}{\delta_{i,j}} A_{i,j} A_{i,j}^T + \delta_{i,j} \frac{N}{8} \sum_i A_{i,j} A_{i,j}^T \right).
\]

Now, for OBD only the diagonal elements of the Hessian are of interest and the saliency becomes

\[
S_{i,j} = \frac{1}{2} \hat{H}_{i,i} W_{i,j}^2 = \frac{N}{8} W_{i,j}^2 \sum_i A_{i,j}^2.
\]

The Hessian elements in Eq. (3) are seen to depend only weakly on the actual location of the weight, i.e., the second term takes a constant value for all elements in a row of the mixing matrix. This may explain in part why the saliency is weakly on the actual location of the weight, i.e., the second order expansion is not always valid. It is well known that the ICA likelihood has measures are compared to the estimated saliency as seen in Fig. 2. There is a comforting agreement, with a tendency towards over-estimation of saliency for OBD and under-estimation for OBD. The largest deviations are found for OBS indicating that the second order expansion is not quite reliable for estimating the retraining configuration, while the cost of deletion is fine for most weights. This, however, does not translate into OBD being better for pruning, because the importance is not directly related to the absolute value of the weight, hence, not an indication of whether a weight is zero in the ‘true’ configuration.

4. Model selection

As we have seen, the pruning procedure is able to produce acceptable results if informed about the number of non-zero elements, hence, the optimal structure is in the nested sequence of pruned models, and we just need to find it. We propose to follow the procedures developed for artificial neural networks and base model selection on either expected generalizability or posterior probability. The posterior probability is aimed at finding the most probably architecture for the given data. However, for finite training samples the most probable architecture need not be the true architecture which generated the data. As the training sample size increases the two measures will converge to make identical decisions.

The posterior probability of the data given the model is found.

\[
p(X|\theta, d) = \int p(X|\theta)p(\theta)d\theta.
\]

If the Laplace approximation is used like in Eq. (1) the following results:

\[
p(X|\theta, d) = e^{-\frac{L_{ML}}{2\kappa}} = e^{-\frac{d}{2\kappa} |H|^{-1/2}},
\]

where only the active (non-zero) variables are included in \( H \), and \( d \) is the number of active variables, i.e., the dimension of \( H \). In the limit of large sample sizes the prior is overwhelmed by data, and therefore the ML estimate is used. This is also the case for non-informative priors. The value has been multiplied by two to better compare to the other measures (as have the training error). The value of \( |H| \) is proportional to \( N^d \) and this approximation is used to form Bayes information criteria [26]:

\[
\text{BIC} = 2L(\theta_{ML}) + d \log N.
\]

Another way of analyzing model fit is to look at the performance on a different set of data, a so-called test set. In the setting of synthetic data, more is simply generated to perform the test. In other cases the available data must be divided, which can be done in different ways.

\[
\text{test} = 2L(\theta_{ML})_{\text{test}}.
\]

Again, the test error has been multiplied by two to compare more easily. By using data that have not been used for training, overfit is avoided and a measure of the generalization error is obtained. The expected value of the test set error can be approximated using Akaike’s information criteria [1]:

\[
\text{AIC} = 2L(\theta_{ML}) + 2d,
\]

which in form is very similar to BIC. The different measures all give an error of which the model with the smallest is preferred. \( L(\theta_{ML}) \) can be both \( L(A_{ML}) \) and \( L(W_{ML}) \).
4.1. Evaluations

In Section 3 the ability to identify the correct zero parameters was tested. In the following the presented model selection methods will be compared to find the number of zero parameters. In principle the four pruning and the four model selection methods can be combined arbitrarily, but, as a starter, MB will be used for pruning. In Fig. 3(A) the different measures are compared for a single run. As seen the ‘training error’ (negative log-likelihood on
7.2 Structure learning by pruning in ICA

As before, we test the ability to find the correct structure as a function of the training set size. We find that BIC is superior for larger data sets. The BIC and the Laplace approximation are both aimed at locating the correct model, hence, more consistent with our evaluation in terms of miss rate than the generalizability measures. The Laplace approximation seems to be too sensitive due to the problems with the second order expansion we have mentioned earlier induced by the singularities in the ICA likelihood, hence, fails more often than the simpler BIC. The failure of the test error can be due to two factors. First, it focuses more on getting the density function correct, hence getting the correct parameters, and second, it is a quite general finding that the cross-validated test error is a noisy quantity. We finally evaluate the efficiency of the different pruning schemes in combination with BIC for model selection. The picture in Fig. 4 confirms our earlier findings: MB, OBS and ARD all perform well for larger data sets. We conclude that the magnitude based pruning in combination with the BIC criteria can provide the correct number of zero elements (Fig. 4A) and that they are also in correct position (Fig. 4B) for sufficiently large sample sizes.

\[ \text{miss rate} = \frac{1}{\text{#experiments}} \sum \left( \sum_{k} z_{\text{est}}^k - z_{\text{true}}^k \right) \]

where \( z = 1 \) to indicate a zero parameter and \( z = 0 \) otherwise, \( K \) is the number of parameters within each experiment.

Fig. 3. Evaluation of the different stop criteria using MB pruning. The 'true' structure is lower triangular as before and has 10 zero parameters out of 25 parameters for the five source model. In figure (A) a single experiment is plotted, each candidate stop criterion locates the correct model. 'Error' means the negative log-likelihood plus relevant penalty terms. In (B) the miss classification rate of the correct number of zeros over 1000 trials is reported.

In Fig. 3(B) we show the miss classification rate of the correct number of zeros over 1000 trials:

\[ \text{miss rate} = \frac{1}{\text{#experiments}} \sum \left( \sum_{k} z_{\text{est}}^k \right) \neq \left( \sum_{k} z_{\text{true}}^k \right) \]

with the BIC criteria can provide the correct number of zero elements (Fig. 4A) and that they are also in correct position (Fig. 4B) for sufficiently large sample sizes.

5. Auditory scene analysis

In order to demonstrate the viability of pruning based sparsification in audio data we consider a blind signal separation scenario involving again five sources and five observations. The set of sources consists of three speech sources, a 'confounding' music piece and a street noise source. The scene consists of two rooms connected by a hallway. The music and the noise sources are considered stationary in position corresponding to a piano and open windows. The speech sources are placed in different locations aimed to mimic movement. The five observations comes from five different microphones placed so that two are located in each room and one in the hallway. The microphones will record from sources in the room where it is placed and from adjacent rooms. This means that the microphones in a given room will record from that room and the hallway but not from the other room. The hallway microphone records everything. Three different placement schemes have been simulated and are illustrated in Fig. 5.

Using the setup from above a mixing matrix is generated based on the distances between sources and observations. The setup in Fig. 5(A) gives a mixing matrix

\[ A = \begin{bmatrix} 0.89 & 0.41 & 0 & 0.21 \\ 0.37 & 0.74 & 0 & 0.30 \\ 0 & 0.32 & 0 & 1.31 & 0.45 \\ 0 & 0.74 & 0 & 0.37 & 0.30 \\ 0.21 & 0.20 & 0.54 & 0.46 & 1.37 \end{bmatrix} \]
As can be seen this matrix has structure, and by identifying the zeros it is possible to allocate sources to the different rooms. The sound is divided into segments of fixed length which will assure stationarity of the positions during a given configuration (A,B,C). The ICA algorithm is run and pruning is performed using MB and BIC. Different signal lengths (sample sizes) are considered to compare performance and the resulting miss classification rates are reported in Fig. 6.

The performance is as before strongly dependent on sample size. For these real signals this is actually more severe due to non-stationarity. For example, the silent parts of speech signals severely limits performance, clearly dramatically so if they span an entire segment. Furthermore, real signals can be colored and they are not
necessarily distributed according to the model assumptions. It can also be seen that the performance is somewhat dependent on the actual placement of the sources. In the first scheme (A) the sources are most evenly placed and this is also the scheme that gives rise to the best performance. The miss classification rate increases in the second and is highest in the third scenario (C) which has most sources located in one room. Despite the higher miss rates, a reasonable performance is achieved for larger samples. If the speakers are not moving too fast it is reasonable to assume stationarity say, within 2–3s then the zero mis-classification rate is approximately 5% in scenarios (A,B).

6. Discussion and conclusion

We have shown that ICA models can be efficiently pruned by pruning. This is highly relevant for causal models, as well as it can assist interpretation for data analytic applications of ICA. We have derived expressions for the Hessians of the likelihood function based on both estimation of mixing and separation matrices. We have shown that the sparsity of a generative model can be recovered with high precision at larger sample sizes. We found that simple magnitude based pruning works well, especially for relatively small samples. This is in contrast to published results for neural networks. This discrepancy may be due to the well-known singularities in the ICA likelihood function induced by co-linearities in the mixing matrix.

To determine the optimal degree of sparsity we recommend using the Bayesian information criterion. This approximation works better than both test sets and the nominally higher order Laplace approximation. Test sets are notoriously noisy, while the problems for the Laplace approximation may again be attributed to the limited validity of Taylor expansions.

Appendix A. Gradients and Hessians for ICA

For training of the model an error function and its gradient is used. For both saliency based pruning and for the ARD approach we also need the Hessian of the negative log-likelihood. We here provide the necessary expressions for parameterization by mixing and separation matrices

\[ \frac{\partial^2 \mathcal{L}(\mathbf{w}, \mathbf{z})}{\partial w_{ij}^2} = NA_{ij} + \sum_i \delta_{ij} \tanh z_i + z_{ij} W_{ij}, \]

\[ = N A_{ij} + \delta_{ij} \tanh z_i + z_{ij} W_{ij}, \]

\[ \frac{\partial^2 \mathcal{L}(\mathbf{w}, \mathbf{z})}{\partial A_{ij}^2} = \frac{\partial^2 \mathcal{L}(\mathbf{w}, \mathbf{z})}{\partial w_{ij}^2} \bigg|_{\mathbf{w} = 0}, \]

\[ \frac{\partial^2 \mathcal{L}(\mathbf{w}, \mathbf{z})}{\partial A_{ij}^2} \bigg|_{\mathbf{w} = 0} \]

References


7.3 Using Gaussian Mixture Models in pruned ICA

Using Gaussian Mixture Models in Pruned ICA.

Andreas Brinch Nielsen, abn@imm.dtu.dk

October 6, 2008

Abstract
This paper expands previous work in structure learning in ICA, by extending the source model, and rigorously testing performance against ordinary ICA. The source model is extended to use a mixture of Gaussians which will be trained as part of the procedure. This allows the algorithm to find sub-Gaussian as well as super-Gaussian sources, which is an extension of the previous method which only allowed super-Gaussian distributions. In the performance analysis it is shown that the accuracy is significantly improved when zeros are present in the mixing matrix, but also shows that for very small data sets, there might not be enough data to achieve this advantage.

1 Introduction

Independent component analysis is a topic of significant current interest in many scientific areas, see, e.g., [6]. ICA is used in a variety of signal processing applications such as reconstruction of sound sources from multi-microphone recordings, image processing, as well as for understanding EEG signals from multiple independent sources in the human brain. ICA has found numerous uses in data analysis ranging from text mining to bioinformatics and modeling of music, see, e.g., the recent conference proceedings [11].

Until recently, emphasis has been on reconstructing the sources as accurately and fast as possible, but more and more attention is given to the structure of the mixing matrix to assist model interpretation [15, 4]. This could be used to identify obscured objects in video [3], or to make causal inference [13, 5]. In general, structure learning reduces complexity and can reduce overfit in short data sequences, hence improves generalizability by Occam’s principle [2].

In [16] a sparse mixing matrix was found using a sparse prior on the parameters, and in [14] the significance of the parameters after training was tested using the Wald statistic. In [9] another approach was taken. Parameters were pruned one by one, and a number of measures was compared for deciding which parameters to prune and when to stop pruning. If parameters are correctly identified as being zero, this has the advantage of not penalizing unpruned parameters, as is the case when using a sparse prior. Further more you have the
advantage of retraining parameters after pruning, which was not possible in the approach of [14].

If the investigated problem has structure in it, there are reasons to identify the zeros. First of all there is the chance of achieving more accurate predictions of the model. If limited amounts of data are present, which is usually the case, the fewer parameters to be estimated the better the estimates will be. In the framework of pruned ICA, you have to estimate the number and placement of zeros, but as we show in this paper you do gain an advantage in accuracy by pruning. Secondly, identifying the structure of the problem allows better interpretation of the data. If the data supports it, you can even draw causal conclusions similar to what they do in [13, 5].

In this paper the performance of pruning the mixing matrix in terms of parameter accuracy is investigated through extensive simulations. In [9] a super-Gaussian source distribution was used, which makes the method unusable for sub-Gaussian sources. In this paper the model is extended to use the Gaussian mixture model, which has the advantage to model a wide range of distributions, also allowing sub-Gaussian sources. Further more, experiments show that an increase in accuracy can be achieved for super-Gaussian sources as well. In section two and three, basics about independent component analysis and the Gaussian mixture model is explained. In section four the method of pruning is shown, and in section five the challenges and chosen solutions are presented. Simulation results are presented in section six. Finally conclusions are included in section seven.

2 Independent Component Analysis

The basic independent component model involves a linear mixture of \( L \) sources into \( L \) observations, \( \mathbf{X} = \mathbf{A}\mathbf{S} \). The only observed variables are \( \mathbf{X} \), and the goal is to find \( \mathbf{A} \) and \( \mathbf{S} \) by assuming the sources to be independent from each other and that they are non Gaussian. \( \mathbf{S} = [\mathbf{s}^1, ..., \mathbf{s}^N] \) is the source matrix, \( \mathbf{X} = [\mathbf{x}^1, ..., \mathbf{x}^N] \) the observation matrix, \( \mathbf{s}^n \) and \( \mathbf{x}^n \) are the source and observation \( L \)-vectors at time \( n \). \( \mathbf{A} \) is a square \( L \times L \) mixing matrix, which is assumed to be invertible, \( \mathbf{W} = \mathbf{A}^{-1} \). Assuming that the source signals are i.i.d., \( p(\mathbf{S}) = \prod_n p(s_n) \) and further more that the sources are independent \( p(s_n) = \prod_i p(s^n_i) \), we obtain the likelihood function,

\[
p(\mathbf{X}|\mathbf{A}) = \prod_n p(\mathbf{x}^n|\mathbf{A}),
\]

\[
= \prod_n \int p(\mathbf{x}^n|\mathbf{s}^n, \mathbf{A}) \prod_i p(s^n_i) ds^n.
\]
Using the separation matrix $W$ the likelihood can be written as,

$$
p(x^n|s^n, A) = \delta(x^n - As^n),
$$

$$
p(x^n|W) = \int \delta(x^n - W^{-1}s^n) \prod_i p(s^n_i) ds^n,
$$

$$
= |W| \prod_i p(\sum_j W_{ij} x^n_j),
$$

where $||$ is the absolute determinant.

We want to fit a model that maximizes the likelihood, which is done by minimizing the negative logarithm of the likelihood,

$$
\mathcal{L}_n = -\log |W| - \sum_i \log p(\sum_j W_{ij} x^n_j).
$$

When taking the logarithm, the product in equation 1 becomes a sum and the derivative is

$$
\frac{\partial \mathcal{L}_n}{\partial W} = -W^{-T} - \gamma^n (x^n)^T,
$$

$$
\gamma^n = [\gamma^n_1, ..., \gamma^n_L]^T, \gamma_i^n = \frac{p'(s^n_i)}{p(s^n_i)},
$$

where $W^{-T} = (W^{-1})^T = (W^T)^{-1}$.

In many cases as well as in [9] a simple parameter free distribution is used, e.g., $p(s) = \frac{1}{\pi} \text{sech}(s)$. The distribution is well suited for many applications including speech and music, but using this distribution only allows to separate super-Gaussian distributions and will fail for sub-Gaussian ones. Extensions of ICA have been proposed to alleviate this drawback; extended ICA [8] uses a parameter to control whether a sub-Gaussian or a super-Gaussian is used for each source, and this parameter can be flipped during training for the individual sources. The approach that is followed here is the same as in independent factor analysis [1], which essentially is ICA using a mixture of Gaussians for the source distributions. The source distributions will be fitted as part of the training, and this allows both super- and sub-Gaussian source distributions.

The error function in equation 2 can also be parameterized using the mixing matrix, $A$, instead of the separation matrix, $W$. In ordinary ICA it does not change the final solutions since the minima are the same, although it can influence convergence properties as investigated in [10]. In the case of pruning it does become important, since zeros in the separation matrix does not readily convert to zeros in the mixing matrix and vice versa. The derivative with regards to the mixing matrix is,

$$
\frac{\partial \mathcal{L}_n}{\partial A} = A^{-T} + A^{-T} \gamma^n (s^n)^T,
$$

where $\gamma^n$ is defined as in equation 3.
3 Gaussian Mixture Model

The Gaussian distribution has many nice properties and is the most common and well investigated distributions of them all. Unfortunately the nice properties come with a price, and the distribution is unsuited for ICA. The reason is that the higher order cumulants are used to separate the sources, and they are all zero for the Gaussian distribution. To take advantage of the many nice aspects of the Gaussian distribution and to allow for more advanced distributions, the Gaussian Mixture Model (GMM) is proposed [2]. As the name suggests the GMM is a sum of a number of Gaussians with individual mean and variance,

\[ p(s) = \sum_k \pi_k \mathcal{N}(s|\mu_k, \sigma_k), \]

where \( \pi_k \) defines the mixture weights (\( \pi_k > 0, \sum_k \pi_k = 1 \)), and \( \mathcal{N}(s|\mu_k, \sigma_k) \) is the Gaussian distribution with mean, \( \mu_k \), and variance, \( \sigma_k^2 \). The Gaussian mixture model can approximate any distribution with arbitrary accuracy, given enough Gaussian components.

3.1 Expectation Maximization

As mentioned the GMM can model any distribution, but of course the value of the parameters of the individual Gaussian components and the mixing weights has to be set. This is done by training the parameters using a training data set, and will be done by maximizing the likelihood of the data. Unfortunately this cannot be done in closed form, and iterative updates can be found by differentiation. First we define the likelihood of a given Gaussian component, \( k \), given a point, \( s^n \),

\[ p(k|s^n) = \frac{\pi_k \mathcal{N}(s^n|\mu_k, \sigma_k)}{\sum_{k'} \pi_{k'} \mathcal{N}(s^n|\mu_{k'}, \sigma_{k'})}, \]

and the effective number of points generated by that component, \( N_k = \sum_n p(k|s^n) \).

Note that the sum of \( N_k \) over all components equals the total number of points, \( \sum_k N_k = N \), and hence the interpretation of \( N_k \) as the effective number of points from a given component. The mean of each component can be found,

\[ \frac{\partial}{\partial \mu_k} \sum_n \log p(s^n) = \sum_n p(k|s^n) \frac{1}{\sigma_k^2} (s^n - \mu_k) = 0, \iff, \]

\[ \mu_k = \frac{1}{N_k} \sum_n p(k|s^n)s^n, \]

Similarly the variance, \( \sigma^2 \), can be found,

\[ \frac{\partial}{\partial \sigma_k^2} \sum_n \log p(s^n) = \sum_n p(k|s^n)(\frac{1}{2\sigma^4}(s^n - \mu_k)^2 - \frac{1}{2\sigma^2}) = 0, \iff, \]

\[ \sigma_k^2 = \frac{1}{N_k} \sum_n p(k|s^n)(s^n - \mu_k)^2. \]
The mixture weights must adhere the constraints, and this can be assured using a Lagrange multiplier,

$$\frac{\partial}{\partial \sigma_k^2} \sum_n \log p(s^n) = \sum_n \frac{1}{\pi_k} p(k|s^n) + \lambda = \frac{N_k}{\pi_k} + \lambda = 0, \Leftrightarrow,$$

$$\lambda \pi_k = -N_k,$$

if this is summed over $k$ on both sides you get $\lambda = -N$ and get the final result,

$$\pi_k = \frac{N_k}{N}.$$

The algorithm is called expectation-maximization (EM) and convergence is assured, but not necessarily to a global maximum. First $p(k|s^n)$ and $N_k$ is computed for fixed Gaussian components (E-step), and then $\mu_k$, $\sigma_k^2$ and $\pi_k$ is found fixing $p(k|s^n)$ and $N_k$ (M-step).

### 3.2 GMM in ICA

In order to use the Gaussian mixture model in independent component analysis, a procedure of training both the independent component model and the Gaussian mixture model must be found. We will use the procedure described by [1] as the seesaw procedure. It works by fixing either the source model or the mixing matrix and training the other. We start by fixing the mixing matrix at the initial guess and finding the source model using the EM procedure described above. With the source model fixed we then train the mixing matrix. Then again the mixing matrix is fixed and the source model is retrained and so forth.

The EM procedure is used as above, but to train the mixing matrix, the derivatives in equation 3 must be found. They are given by,

$$\frac{p'(s^n_i)}{p(s^n_i)} = - \sum_k p(k|s^n_i) \frac{1}{\sigma^2_{ik}} (s^n_i - \mu_{ik}).$$

### 4 Pruning parameters

The goal of this paper is to find the structure of the mixing matrix in an independent component analysis problem. By the structure is meant the number and placement of zeros in the mixing matrix, which can be interpreted as finding the active links between sources and observations in a fully connected network. The procedure of finding structure will be that of backward elimination of links. This means we start off in a fully connected network and selectively prune out links (parameters). Two questions need answering when pruning parameters - which parameter to prune next and when to stop.

In [9] a number of different approaches were compared for obtaining the saliency of the individual parameters in the mixing matrix. It was found that
using the magnitude won the competition against second order methods, probably because of the very nonlinear behavior of the error surface in ICA. Further more the simple BIC measure was superior to other more advanced measures such as Laplace approximations and computing test errors.

Using the magnitude as the pruning criterium is probably the most simple criterium possible. When the model is trained the absolute value of the parameters are compared and the smallest is the next in line to be pruned. Even though the magnitude is the simplest it does make good sense in ICA, since the sources are linearly dependent on the parameters, and both observations and sources are normalized to have unit variance.

4.1 Bayesian Information Criterium

The posterior probability of the data given the model is found,

\[ p(X|M) = \int p(X|\theta)p(\theta)d\theta. \]

Even when assuming a non-informative prior \(p(\theta) = k\), this integral cannot be solved directly for the ICA problem.

Assuming a non-informative prior, the log likelihood can be approximated using a second order Taylor expansion around the maximum likelihood (ML) estimate of the parameters. This is equivalent to approximating the posterior distribution by an unnormalized Gaussian distribution and is called the Laplace approximation. Because the ML estimate is a maximum, the first order derivative vanishes,

\[ p(X|M) \approx \int e^{-L(\theta_{ML})-\frac{1}{2}\Delta^T H \Delta} \Delta \theta, \]

\[ = e^{-L(\theta_{ML})} \frac{1}{2} \Delta^T H^{-1} \Delta, \]

\[ -\log p(X|M) \cong L(\theta_{ML}) + \frac{1}{2} \log |H| - \frac{d}{2} \log 2\pi. \]

\[ \Delta = \theta_{ML} - \theta, \quad H = \frac{\partial^2 L(\theta)}{\partial \theta \partial \theta^T} \quad \text{and} \quad || \text{is the absolute determinant. Only the active (non zero) variables are included in } H, \text{ and } d \text{ is the number of active variables, i.e. the dimension of } H. \]

\[ H \text{ consists of a sum over samples and by making this relation explicit you observe that } |H| \text{ is proportional to } N^d. \text{ Ignoring constant terms and the proportionality factor of } |H| \text{ you get the following approximation,} \]

\[ -2 \log p(X|M) \cong 2L(\theta_{ML}) + d \log N = BIC, \]

which is called Bayes Information Criterium [12]. The model with the lowest BIC value is the preferred model.
5 Challenges

The model is likely very good if trained properly, but a number of issues is present in ICA, in the pruning of parameters in ICA, and especially in using the Gaussian mixture model for source distributions. All of these challenges occur in the training phase of the method, they will be explained in the following together with the ways we try to alleviate them.

5.1 Ambiguities in ICA

The solution to the ICA problem is never unique in the sense that other solutions exist that are equally valid. This is because there is inherent ambiguities build into the model; one is the scaling ambiguity and the other is the permutation ambiguity.

As the sources are unknown so are their distributions and therefore their variance. This means a scaling of the mixing matrix can be completely absorbed by a similar, inverse, scaling of the sources,

\[ X = AS = ADD^{-1}S, \]

where \( D \) is a diagonal scaling matrix. The scaling ambiguity is usually dealt with by assuming a fixed variance of the sources.

The other ambiguity that exist for ICA is the permutation ambiguity. Again since we know nothing about the sources we do not know in which sequence they occur. A permutation of the sources can be exactly absorbed by a similar permutation of the columns of the mixing matrix and produce the same observations,

\[ X = AS = APP^{-1}S, \]

where \( P \) is a permutation matrix. The permutation problem cannot be dealt with, but rarely poses a problem, since the labeling of sources as one, two, ... is not a major concern and can be changed if necessary.

In ordinary ICA neither of the ambiguities poses real problems since the different solutions has the same properties and in essence is a matter of scaling and labeling the sources. If this is of importance it must be based on postprocessing based on additional knowledge about the content of the sources. In the pruning case it is not quite so clear if the ambiguities pose a problem, especially because the pruning is done in a iterative way. A problem comes about when forcing a parameter to zero. In theory the ICA algorithm is able to repermute the sources during retraining after a parameter has been set to zero, effectively setting another parameter to zero. Whether this is a problem is hard to say, and will show in the experiments. Since the magnitude based pruning is used, the parameters which are forced to zero are already close to zero, and therefore we do not expect it to be a problem.
5.2 GMM considerations

The Gaussian mixture model can model any distribution and therefore has arbitrary variance. This introduces some problems, since the determinant of the mixing matrix is part of the optimization of the parameters. The variance will be forced to one during each optimization of the GMM. The variance of a GMM can be found like this,

\[
\begin{align*}
\mu &= \int p(s)sd s = \int \sum_k \pi_k \mathcal{N}(s|\mu_k, \sigma_k)sd s = \sum_k \pi_k \mu_k, \\
\sigma^2 &= \int p(s)(s - \mu)^2ds = \int \sum_k \pi_k \mathcal{N}(s|\mu_k, \sigma_k)(s - \mu)^2ds, \\
&= \sum_k \pi_k (\sigma^2_k + \mu^2_k) - \mu^2.
\end{align*}
\]

After finding the parameters of the GMM, the mean and variances are updated by dividing by the square root of the variance found above. The mixing or separation matrix is updated too, to make the sources and distributions fit each other. Each column of the mixing matrix is multiplied by the square root of the variance or equivalently each row of the separation matrix is divided by the square root of the variance.

Singular solutions can occur if a Gaussian component collapses onto a single data point. In this situation the EM algorithm will assign variance close to zero, which gives infinite log likelihood and thus is optimal in the maximum likelihood sense, but obviously is very bad for generalization. This is avoided by setting a lower limit on \(\sigma_k\). The GMM distribution is continuously normalized to have unit variance and therefore a fixed lower limit is possible, \(\sigma_k > 0.1\).

The GMM does not have a well defined global maximum, because permutations of the global maximum exist. These can be found simply by permuting the components, obviously leading to the same probabilities. This is not a real issue since we do not care which permutation we get. Another problem is that local maxima exist, and practice has shown that convergence results depend heavily on the initialization of the algorithm. This can be helped by training multiple models and selecting the best.

5.3 Getting stuck in ICA

If you have bimodal source distributions you can expect to see observations as the one in figure 1. The Gaussian mixture model can model this if enough components (four or more) are used. If the model fits the observation tightly the update of the mixing matrix is hampered because an update causes a shift of some of the modes, which, because of the small width of each mode, is very unlikely given the source distribution.

To circumvent the problem some strategies are followed. If the mixing matrix is initialized very close to the true mixing matrix, the first source distribution will be close to the final source distribution and it is likely that the initial sources
do not have more modes than the true source distribution, thus avoiding the
overfit of the model. This initialization is done using FastICA [7]. The FastICA
algorithm can handle both sub- and super-Gaussian source distributions and
can therefore be used to generate the initial mixing matrix. Another approach
to limit the bad effects is to limit the number of components of the source dis-
tributions to a relatively low number. This way the number of modalities is
limited and therefore over fitting is reduced. This also helps keeping computa-
tional time down because it saves the time consuming model order estimation in
the Gaussian components. Three components for each Gaussian mixture model
is used throughout the paper. A third and last approach is to have the floor
on the variance on each component be rather high. Usually the floor is selected
to avoid computational problems with division of zero. As was stated in the
previous subsection the floor is set to 0.1. This way the modes can always be
moved a little to the sides with out exploding the error.

6 Simulations

If limited data is available the more parameters to estimate will increase the
variance of the estimations, which is a consequence of the bias-variance trade
off. In theory you should get better estimates of the parameters if you limit the
number to be found.

Whether better estimates is achievable is investigated in this section, and
to have control over the experiments it will be done using synthetic data. To
give as general a picture as possible many experiments have been performed
with different settings. Source distributions and mixing matrices are generated,
so that exact comparisons and evaluations can be made. Every experiment is
repeated a 500 times and median errors are reported.

Four models will be compared. The standard ICA model using the sech distribution will simply be called ICA, and if pruning is used it will be called pICA. Using the Gaussian mixture model we will call it GMICA and pGMICA for the unpruned and pruned case respectively.

6.1 Experimental settings

Since this is a simulated data set we have the true mixing matrix. The reported error is therefore the squared error between the estimate and the true. To compute this, scaling and permutation ambiguity has to be solved, and this is done by comparing the true sources to the found sources and maximizing the squared error between them by permuting and scaling the mixing matrix. When the correct permuted and scaled mixing matrix is found the mixing matrix squared error is computed,

$$\text{MMSE} = \sum_{i,j} (\hat{W}_{ij} - W_{ij})^2,$$

where $W_{ij}$ are the true matrix parameters, and $\hat{W}_{ij}$ are the estimates.

Different distributions are used. The Gaussian distribution forms the basis and different exponentials are taken keeping the sign,

$$x = \mathcal{N}(0, 1), \tilde{x} = |x|^{\alpha} \text{sign}(x).$$

For $\alpha$ larger than one a super-Gaussian distribution is obtained, and smaller than one is sub-Gaussian. Four different assignments of $\alpha$ have been used, \{0.1, 0.8, 1.2, 1.8\}. These cover two sub-Gaussian and two super-Gaussian distributions. A final experiment was performed for a four source case, where each component had its own distribution.

In [9] it was found that pruning performance was dependent on the number of samples in the data set. Therefore every experiment will be performed using different numbers of samples. The range is from [100, 5000] samples which was found to cover both very poor and very good performance. The range is divided in sort of exponential increasing numbers to have a better resolution in the small sample sizes than in the larger sizes.

We test on different number of sources, three and four, to illuminate if there is a dependence on this. Because the algorithm is time consuming especially when repeated as many times as it is here, only three and four dimensions are considered, but limited experiments have been carried out that give the same conclusions as is presented here.

The number of zeros has the potential to impact performance as well. Especially it would be interesting to see if the framework harms performance in the case of no zero parameters. The algorithms are run with the number of zeros ranging from zero to ten in the separation matrix.
Figure 2: Mixing matrix squared error (MMSE). (A) three sources, (B) four sources. Number of zeros indicated in legend. Super-Gaussian source distributions (exponent of 1.2). For all but very small sample sizes and small number of zeros, there is an increase in accuracy if the pruning is used. This is the case for both three and four sources. If no zeros are in the true separation matrix the performance is hurt for small sample sizes. The more zeros that are in the mixing matrix, the more is gained from pruning ending with more than a ten fold decrease in MMSE.

6.2 Results

In this section an overview of the observations about accuracy will be given. The observations are illustrated in figures, and their captions contain further details.

The first experiments, figure 2, compares ICA versus pICA. The squared error is improved in all cases but for few samples together with no zeros. For many zeros and enough samples more than a ten fold decrease in error is achieved.

In figure 3 (A) ICA and GMICA are compared. Quite interestingly even in the event of super Gaussian data the GMICA shows to be superior. For very small sample sizes ICA performs better though. The GMICA algorithm introduces extra parameters to be estimated in the source distributions and it seems that additional samples are needed to get consistent estimates. The same comparison is done for the case where pruning is used, and the same conclusions apply. For small sample sizes pICA is better because of fewer parameters to be fitted, but for larger sample sizes pGMICA is better because of the superior modeling capabilities.

This aspect is inspected further in figure 4 where the true source distribution is plotted together with the sech distribution and a fitted Gaussian mixture model using three components as in the experiments. Clearly the Gaussian mixture model is able to model the distributions closer, both in the sub-Gaussian cases and in the super-Gaussian cases.

In figure 5 the comparison between GMICA and pGMICA is shown. For very small sample sizes and one zero, the accuracy is better for the unpruned
7.3 Using Gaussian Mixture Models in pruned ICA

Figure 3: Mixing matrix squared error (MMSE). (A) unpruned, (B) pruned. Four sources. Zeros indicated in legend. Super-Gaussian source distributions (exponent of 1.2). Using the Gaussian mixture for source distributions helps precision, which is probably because the GMM is a closer fit for the exponentiated Gaussian, than is the sech distribution. The decrease in MMSE is also active when pruning is used. For both cases the decrease is only present when "enough" samples are available, which is a sign that more samples are needed in order to get reliable estimates of the source distributions.

Figure 4: The four different source distributions and how the two models fit them. Exponentiated (true) distribution in dotted, Gaussian mixture model in full and sech is dashed. From left to right $\alpha$ equals $[0.1, 0.8, 1.2, 1.8]$. The Gaussian mixture model has been fitted on 10000 data points generated using the appropriate distribution. Obviously the Gaussian mixture model is able to fit the sub-Gaussian distributions better than sech, but also for the super-Gaussian distributions the fit is closer.
Figure 5: Mixing matrix squared error (MMSE). (A) Three sources and (B) four sources. Zeros indicated in legend. Super-Gaussian source distributions (exponent of 0.8). In the case of GMM it is still better to use pruning than not. For small sample sizes and one zero the performance is a bit worse than without pruning. This is because not enough samples are available to make a reliable estimate of the placement and number of zeros.

Figure 6: (A) Four sources and one zero. The exponent is indicated in the legend. It is clear that the sech distribution is not valid in the case of sub-Gaussian sources. The GMM model shows good performance for all cases. Note that the performance is best for $\alpha = 0.1$ and $\alpha = 1.8$. This is because the ICA algorithm uses non-Gaussianity, and the longer away from Gaussianity the better the solution. (B) Four sources. Each source has a different source distribution using all of the possible $\alpha$ values. Again the hyperbolic secant source distribution is insufficient and only the GMM can solve the problem.
case, but for the other cases the performance is better when pruning, and again more than a ten fold decrease in error is achieved when enough samples are available.

In figure 6 (A) pGMICA and pICA are compared in the setting of each distribution. Clearly the sech distribution fails in the sub-Gaussian cases, but even in the super-Gaussian cases it is only for the small sample sizes that it can keep up with the Gaussian mixture model. In all other cases the pGMICA is superior by quite a margin. An interesting observation is that the pGMICA works better in the case of $\alpha = 0.1$ and $\alpha = 1.8$, than in the other two cases. The explanation lies in the fact that ICA lives off the non-Gaussianity of the data, and the further away from $\alpha = 1$ the better for the model. This is the case for both GMICA and ICA.

As a final experiment all of the possible source distributions is used at the same time, see figure 6 (B). Again pICA fails to capture the sub-Gaussian parts, and fails in finding the parameters. The pGMICA has no problem in this setting and this shows the superiority of this model. It can work for arbitrary combinations of non-Gaussian sources.

The general observations are that only for few samples and no or few zeroes it is worse to use the pruning algorithm. When comparing the two source distributions the choice is obvious for sub-Gaussian sources since ICA breaks down completely in this case. For super-Gaussian sources ICA is to be preferred for small sample sizes, and if computational simplicity is a concern. For other cases pGMICA improves results.

7 Conclusion

Extensive experiments were performed focusing on the increase in accuracy by pruning the mixing/separation matrix in independent component analysis. It was found that for reasonable sample sizes or for more than one zero there was a significant increase in accuracy. Additionally the original model was extended to use the Gaussian mixture model. This allows a wider range of source distributions and particularly it allows the separation of sub-Gaussian sources. The same increase in accuracy was observed when pruning this model. Further more it was found that the Gaussian mixture model also caused an increase in accuracy for super-Gaussian sources compared to the original model using the sech distribution. This is most likely because the Gaussian mixture model allows a tighter fit of the source distribution.

Using the Gaussian mixture model allows a detailed modeling of the source distributions and the pruning model allows specific models to be trained. This has the possible applications of comparing different causal networks in a proper likelihood setting, which is not possible with earlier suggested sparse ICA models.
References


Chapter 8

Conclusion

The general topic of the project have been the extraction of information from audio. This have been achieved through the extensive use and development of various machine learning techniques, and by taking two different approaches to the modeling of audio. The first approach focused on single channel observations and classification of different aspects of audio centered around the use of pitch dynamics. The other approach have been through structure investigations of mixings of multiple sources.

In the audio classification domain it has been shown that pitch is a very relevant feature for identifying various information. Pitch is an interesting feature since it is robust to different ways of processing and can be computed quite efficiently. It was used to classify audio into three classes, music, noise and speech, and a quite high accuracy rate of 5 % was achieved on a very diverse database of audio. One interesting aspect of this is that not only pitch was used, but also the level of pitchness have relevance and expands the area of use of pitch to signals that are not harmonic. Two models of musical instruments were compared and a fixed envelope model was found to be the most accurate for the inspected range of musical instruments which covered many classical instruments. Further more a method of segmenting popular music into voiced and unvoiced pieces was investigated. In this investigation it was revealed that large databases of music is necessary because the performance is dependent on artists and if there is an overlap in artists between training and test sets.
Conclusion

Classification of whether two recordings was from the same environment was done with high accuracy. Two methods were compared and a using a combination of them both a fast an reliable algorithm was found.

Pruning of mixing parameters in a linear model of observations was used to obtain greater accuracy of the estimates. It was shown that the same data can be used to identify the number of zero parameters as well as the estimation of the remaining parameter values. Furthermore, the structure that is revealed can be used in the interpretation of the model, and might allow for causal hypotheses between the observations. It was achieved through Independent Component Analysis and pruning of the mixing matrix. Two models were proposed. The first was the classic model using a fixed super-Gaussian source distribution, which is quite suitable for many audio applications. The second model used the Gaussian Mixture Model in the source distributions, which allows a much wider set of source distributions, and opens the use of the model to more applications. The advantage of the proposed methods over previously suggested methods, is the probabilistic framework in which it is developed. This allows for the general evaluation of model fit which is used to prune the parameters away.

8.1 Future work

The studies done in this project only scratch in the surface of proactive computing and much work remains to be done. In general, classification of various aspects of audio is still open, but especially robustness to noise needs to be investigated. Pitch could be an important part of this work because of its invariance to noise, but there is the concern that you might be pushing the problems into the pitch estimation instead. Another goal of future research is the ability to classify mixtures of classes, which could be speech in noise, or multiple instruments playing all at once. Such settings might be handled by blind signal separation algorithms prior to classification.

In the structure learning domain the introduction of Independent Component Analysis is promising because it is a better assumption than uncorrelatedness. One concern is if the square linear models are too limiting to present the causal relations between observations. This opens the question if non-square models can be solved and if the techniques can be extended to nonlinear methods.
In this appendix the details of the derivation of the gradient equations are reviewed. First the definitions of the network are stated.

\[
E = - \sum_{n=1}^{N} \sum_{k} t^n_k \log \hat{t}^n_k,
\]

\[
\hat{t}^n_k = \frac{e^{y_k}}{\sum_{k'=1}^{K} e^{y_{k'}}},
\]

\[
y^t_k = \sum_{j=0}^{J} w_{jk} h^t_j,
\]

\[
h^t_j = \tanh(u^t_j),
\]

\[
u^t_j = \sum_{i=0}^{I} w_{ij} x_i.
\]

$I$ is the number of inputs, $J$ is the number of hidden units and $K$ is the number of outputs. In general $i$ indexes the inputs, $j$ the hidden units and $k$ the outputs. $x$ is the input, $u$ is the input to the hidden units, $h$ the output of a hidden unit, $y$ is the output of the network, and $\hat{t}$ is the output of the output function.

The derivative of the error function compared to the output weights are found
like this,

\[ \frac{\partial y^n_k}{\partial w_{jk}} = h^o_j, \]

\[ \frac{\partial \hat{t}_k^n}{\partial y^n_k} = \frac{\delta_{kk'} e_{y^n_k} \sum_{k''=1}^K e_{y^n_{k''}} - e_{y^n_k} e_{y^n_{k'}}}{\left( \sum_{k''=1}^K e_{y^n_{k''}} \right)^2}, \]

\[ = \frac{\delta_{kk'} e_{y^n_k} \sum_{k''=1}^K e_{y^n_{k''}}}{\left( \sum_{k''=1}^K e_{y^n_{k''}} \right)^2} - \frac{e_{y^n_k} e_{y^n_{k'}}}{\left( \sum_{k''=1}^K e_{y^n_{k''}} \right)^2} = \delta_{kk'} \hat{t}_k^n - \hat{t}_k^n \hat{t}_{k'}^n, \]

\[ \frac{\partial E}{\partial w_{jk}} = \sum_{n=1}^N h^n_j (\hat{t}_k^n - t_k^n), \]

\[ \frac{\partial E}{\partial w_{jk}} = - \sum_{n=1}^N \sum_{k'=1}^K \frac{1}{t_k^n} \frac{\partial \hat{t}_k^n}{\partial y^n_k} \frac{\partial y^n_k}{\partial w_{jk}}, \]

\[ = - \sum_{n=1}^N \sum_{k'=1}^K t_k^n \frac{1}{t_k^n} (\delta_{kk'} \hat{t}_k^n - \hat{t}_k^n \hat{t}_{k'}^n) h^o_j, \]

\[ = - \sum_{n=1}^N \sum_{k'=1}^K t_k^n \frac{1}{t_k^n} \left( \sum_{k''=1}^K \frac{1}{t_k'^n} \delta_{kk''} \hat{t}_k^n - \sum_{k''=1}^K \frac{1}{t_k'^n} \hat{t}_k^n \hat{t}_{k''}^n \right), \]

\[ = - \sum_{n=1}^N \sum_{k'=1}^K t_k^n \frac{1}{t_k^n} = \sum_{n=1}^N h^n_j (\hat{t}_k^n - t_k^n). \]

The derivative of the error function compared to the input weights are found like this,

\[ \frac{\partial y^n_{k''}}{\partial w_{ij}} = w_{jk''} \frac{\partial h^n_j}{\partial w_{ij}}, \]

\[ \frac{\partial h^n_j}{\partial w_{ij}} = (1 - (h^n_j)^2) \frac{\partial u^n_i}{\partial w_{ij}}, \]

\[ \frac{\partial u^n_i}{\partial w_{ij}} = x^n_i, \]

\[ \frac{\partial E}{\partial w_{ij}} = - \sum_{n=1}^N \sum_{k'=1}^K \frac{1}{t_k^n} \sum_{k''=1}^K \frac{\partial \hat{t}_k^n}{\partial y^n_{k''}} \frac{\partial y^n_{k''}}{\partial w_{ij}}, \]

\[ = - \sum_{n=1}^N \sum_{k'=1}^K \frac{1}{t_k^n} \sum_{k''=1}^K \left( \delta_{kk''} \hat{t}_k^n - \hat{t}_k^n \hat{t}_{k''}^n \right) w_{jk''} (1 - (h^n_j)^2) x^n_i, \]

\[ = \sum_{n=1}^N (1 - (h^n_j)^2) x^n_i \sum_{k'=1}^K \left( \hat{t}_k^n - t_k^n \right) w_{jk'}, \]
The increase in error, $\Delta E_i \approx S_i$ is to be optimized with the constraint that $e_i^T \Delta w + w_i = 0$. This can be done using a Lagrange multiplier,

$$\frac{\partial}{\partial \Delta w} \left( \frac{1}{2} \Delta w^T H \Delta w + \lambda \left( e_i^T \Delta w + w_i \right) \right) = 0, \Leftrightarrow \ \ \ \ H \Delta w + \lambda e_i = 0, \Leftrightarrow \ \ \ \ \Delta w = -\lambda H^{-1} e_i.$$

$$\frac{\partial}{\partial \lambda} \left( \frac{1}{2} \Delta w^T H \Delta w + \lambda \left( e_i^T \Delta w + w_i \right) \right) = 0, \Leftrightarrow \ \ \ e_i^T \Delta w + w_i = -e_i^T \lambda H^{-1} e_i + w_i, \Leftrightarrow \ \ \ \ \lambda = \frac{w_i}{(H^{-1})_{ii}},$$

$$\Delta w = -\frac{w_i}{(H^{-1})_{ii}} H^{-1} e_i.$$

Using the found parameter, the increase in error under the constraint can be found to be,

$$\Delta E_i \approx S_i = \frac{1}{2 \left( H^{-1} \right)_{ii}} e_i^T H^{-1} H H^{-1} e_i \frac{w_i}{(H^{-1})_{ii}},$$

$$= \frac{1}{2 \left( H^{-1} \right)_{ii}}.$$
Bibliography


