Université de Montréal

Automatic music classification using boosting algorithms and auditory features

par
Norman Casagrande

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de Maîtrise de Sciences (M.Sc.)
en informatique

Octobre, 2005

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé:

Automatic music classification using boosting algorithms and auditory features

présenté par:

Norman Casagrande

a été évalué par un jury composé des personnes suivantes:

Guy Lapalme
président-rapporteur

Douglas Eck
directeur de recherche

Balázs Kégl
codirecteur

Yoshua Bengio
membre du jury

Mémoire accepté le 22 décembre 2005

RÉSUMÉ

Avec l'augmentation de l'accessibilité à la musique digitale, il devient de plus en plus nécessaire d'automatiser l'organisation de bases de données musicales. Actuellement, les utilisateurs organisent leur bases de données en se basant sur les méta-données des pièces musicales, telles que le nom de l'interprète, le titre de la pièce et de l'album, en plus d'aspects plus subjectifs comme le genre musical et l'ambiance. Par contre, ces informations doivent normalement être ajoutées manuellement et ne sont souvent pas précises ou complètes.

Dans ce mémoire, nous présentons plusieurs techniques d'extractions de caractéristiques ("features") de fichiers audio qui sont utiles pour la classification de pièces musicales dans certaines catégories définies par l'usager. Ces techniques utilisent des concepts la théorie de traitement de signal, de la psychoacoustique et de la perception auditive humaine.

Pour traiter la grande quantité de charactéristiques générées par notre approche, nous utilisons une version multi-classe de l'algorithme AdaBoost. Nous décrivons cet algorithme et nous le comparons avec d'autres approches. Nous présentons aussi des expériences tendant à montrer que notre méthode est plus performante que d'autres méthodes de pointe sur une collection musicale étiquetée manuellement pour la classification de genres musicaux. Nous analysons aussi la performance de notre système à la compétition internationale MIREX 2005, où notre système a terminé premier parmi les 15 soumissions.

Finalement, nous présentons une nouvelle approche pour la modélisation de la structure de la voix, utilisant une représentation visuelle de celle-ci. Nous montrons que ce modèle peut déterminer si un signal sonore correspond à de la voix ou à de la musique, et ce en utilisant une fraction des ressources CPU qui sont nécessaires aux autres approches publiées.

Keywords: AdaBoost, apprentissage machine, recherce d'information musicale, classificatoin de genres musicaux, distinction de la voix et la musique, classification multi-classe, similarité musicale

ABSTRACT

With the huge increase in the availability of digital music, it has become more important to automate the task of organizing databases of musical pieces. Currently users have to rely on meta-information such as performer, album, title, as well as more subjective aspects such as genre and mood, to arrange these databases or to create playlists. However, this information must be added manually and it is often inaccurate or incomplete.

In this thesis we describe how extract from the audio files features that are meaningful for a classification into categories defined by the user. We show how to extract information using knowledge from signal processing theory, physics of sound, psychoacoustics, human auditory perception, and use it to discriminate high-level characteristics of music, such as music genre.

To deal with the very large number of features generated by our approach, we use a multiclass version of the ensemble learner AdaBoost. We describe our algorithm and compare it to other approaches. We present experimental evidence that our method is superior to state-of-the-art algorithms on a hand-labeled database for genre recognition, and describe our performance at the recent MIREX 2005 international contest in music information retrieval, where our algorithm was the best genre classifier among 15 submissions.

We also present a novel approach to model the pattern of specific sounds (i.e. speech), using a visual representation of sound. We show how this model can successfully discriminate speech from music at the frame level, using a fraction of the CPU usage of other approaches.

Keywords: AdaBoost, machine learning, music information retrieval, audio genre classification, music/speech discrimination, multi-class classification, music similarity, automatic playlist generation

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

NOTATION

General:

$\mathbb{R}$	Real numbers
n	The number of examples
D_n	The training set of size n
k	The number of classes
$\mathbf{x}_i$	The vector of observation for example i, where $\mathbf{x}_i \in \mathbb{R}^d$
d	The number of elements (or dimensions) of the observation vector, that is $d = \lvert\mathbf{x}\rvert$
$x^{(j)}$	The jth element of vector $\mathbf{x}$
y_i	The label of example $\mathbf{x}_i$ for binary classification, where $y_i \in \{-1, 1\}$
$\mathbf{y}$	The vector of labels, for multi-class calssification
ℓ	The label index
$y_{i,\ell}$	$\begin{cases} 1 & \text{if } \mathbf{x}_i \text{ belongs to class } \ell \\ -1 & \text{otherwise.} \end{cases}$
δ	An arbitrary small value bigger than zero
$\mathbb{I}\{x\}$	$\begin{cases} 1 & \text{if } x = \textbf{true} \\ 0 & \text{otherwise} \end{cases}$
$x \triangleq y$	x is defined as y

Binary AdaBoost:

$\alpha^{(t)}$	The confidence parameter at iteration t, where $\alpha \in \mathbb{R}$
$\mathbf{w}$	The real valued vector of weights
w_i	The weight of example i, where $w_i \in \mathbb{R}$
t	The iteration index
T	The total number of iterations

$h^{(t)}(\mathbf{x})$ The *hypothesis function*, found by the *weak learner* at iteration t, where for our problems $h : \mathbb{R}^d \to \{-1, +1\}$

$\mathcal{H}$ The space of the hypotheses ($h \in \mathcal{H}$)

$f^{(T)}(\mathbf{x})$ The strong learner, defined as $\sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\mathbf{x})$, where $f : \mathbb{R}^d \to \mathbb{R}$

$\epsilon^{(t)}$ The error, defined as $\sum_{i=1}^{n} w_i^{(t)} \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) \neq y_i\right\}$

$\gamma^{(t)}$ The *edge*, defined as $1 - 2\epsilon^{(t)}$

ρ_i The unnormalized functional margin of example i, defined as $f^{(T)}(\mathbf{x}_i) y_i$

$\hat{R}$ The misclassification rate

$\hat{R}_L$ A generic (based on a loss function L) empirical risk

$\hat{R}_e$ The exponential empirical risk

$Z^{(t)}$ The normalization factor to keep the weights a distribution. Is defined as $\sum_{i=1}^{n} \left(w_i^{(t)} \exp\left(-\alpha^{(t)} h^{(t)}(\mathbf{x}_i) y_i\right)\right)$

AdaBoost Extensions:

ϵ_+ The sum of weights for *correctly* classified examples

ϵ_- The sum of weights for *incorrectly* classified examples

ϵ_0 The sum of weights for *abstained* examples

θ The edge offset parameter, where $\theta \in \mathbb{R}$

Decision Stumps:

q The threshold of the cut, where $q \in \mathbb{R}$

v The alignment value, where $v \in \{-1, 1\}$

$$\varphi_{j,q}(\mathbf{x}) \quad \begin{cases} 1 & \text{if } x^{(j)} \geq q, \\ -1 & \text{otherwise} \end{cases}$$

AdaBoost.MH:

$w_{i,\ell}$ The weight of example i and class ℓ

$\mathbf{v}$ The alignment vector, where $v \in \{-1, 1\}^k$

$\mathbf{g}(\mathbf{x})$ The multi-class strong learner, where $\mathbf{g} : \mathbb{R}^d \to \mathbb{R}^k$

AdaBoost.MO:

$\mathcal{Y}$	The set of the labels
$H(\mathbf{x})$	The k-polichotomy, that is $H : \mathcal{X} \rightarrow \mathcal{Y}$
$\lambda(y)$	The function that maps a label y into one of the two complementary sets $\mathcal{Y}^+$ or $\mathcal{Y}^-$
$x\Delta y$	The Hamming distance between x and y

Features:

$\mathbf{u}$	A real valued vector of length T that represent either a frame (on frame-level features) or the whole signal (on autocorrelation)
τ	The length of the signal, that is $\tau = \|\mathbf{u}\|$
dct	The discrete cosine transformation
$\mathbf{F}$	The discrete Fourier transformation (DFT) of frame $\mathbf{u}$
$\mathbf{a}$	A real valued vector representing the autocorrelation
l	The lag index
A	A real valued matrix representing the phase autocorrelation
ϕ	The phase
$S(\mathbf{x})$	The Shannon entropy of input vector $\mathbf{x}$, where $S : \mathbb{R}^{\|\mathbf{x}\|} \rightarrow \mathbb{R}$
$\Sigma^{(x,y)}$	The integral image at point (x, y) of image I, defined as $\sum_{i=1}^{x} \sum_{j=1}^{y} I^{(i,j)}$

To my family.

ACKNOWLEDGEMENTS

I would like to thank my supervisors, Dr. Douglas Eck and Dr. Balázs Kégl, for their help, encouragement and support. Thanks also to James Bergstra who shared with me most of the work on genre recognition, and Dumitru Erhan for running the experiments with SVMs. Thanks to all my friends at the GAMME and LISA laboratories at the Université de Montréal for the wonderful times and interesting conversations.

Thanks to my family, in particular to my mother, father and sister for supporting me all these years. Thanks to my girlfriend for the constant support and for keeping my motivation up in the last two years.

CHAPTER 1

INTRODUCTION

In the last two decades digital music has transformed the landscape of music distribution. Originally developed by Philips for its Compact Discs, this technology with the advent of the internet has reached its true potential. Once digitalized, a song can be copied millions of times without any production cost, and through the internet immediately distributed into the home of the customer. A successful example of this revolution is the iTunesMusicStore, from Apple, which sold more than 500 millions files since it was launched in 2003.

Portable MP3 players also play a key role, by eliminating the physical support that was once associated with music, allowing large databases of music (an iPod can contain up to 60 Gigabytes of data) to be carried anywhere.

This increased storage, coincided with the parallel increase in the ways that a user can access and purchase online music. The music store from Apple has a catalog of more than 2 million songs, which is, on its own, several orders of magnitude smaller of what is available on P2P networks. This wide selection is not the result of an increase in production of music, but instead the effect of the two key factors that we have previously mentioned: ease of distribution and copying.

As a result, it is becoming increasingly harder to arrange and navigate the large databases of music that are available to the user. Currently, the majority of tools for organizing music use naive grouping strategies based on meta-data information introduced by the user or the creator. Short text fields describe the performer, composer, album, title, as well as more subjective aspects such as genre and mood. Unfortunately this meta-data is often inaccurate or incomplete. Projects like MusicBrainz (`musicbrainz.org`) promise to address this issue by analyzing each file to obtain a fingerprint, which it used to query a central database for meta-data. However, even assuming a complete meta-tag data, there is still a problem with relying on a canonical set of moods or genres to be used by all listeners. In fact,

although researchers have defined full taxonomies for the purpose of testing model performance (e.g. MacKay and Fujinaga [MF05]), there is still no canonical taxonomy of record music, nor will there ever be. Mood has the same problem, even if some attempts have been made in the past to take advantage of it [LLZ03].

Because of these problems, building playlists or discovering new music using meta-tags is extremely hard. Ideally the user should be able to submit a set of music files ("keysongs", instead of keywords) and obtain in return a list of similar entries. This would allow the listener both to dynamically build playlists using his database, and intelligently explore the vast archive of music online.

Recently there has been many efforts to makes this scenario a reality. Software such as Indy (`indy.tv`) or iRate (`irate.sourceforge.net`) rely on collaborative filtering techniques to create playlists based on the interests of a user by collecting taste information from a network of listeners. The basic idea is that if in the past two users have agreed, they will also tend to agree in the future. This approach allows a user to discover new content, but suffers from three key disadvantages. The first is that if the pool of users is too small in comparison to the number of songs, there will be songs in the user's database that cannot be used to draw a profile, because nobody but the user has ever listened to them. This gives rise to the second problem: if there is a new band which is proposing its work, it is not possible to position it in the "preference" space, as nobody knows anything about it yet. Finally, the preference given to a music piece may greatly vary depending on the situation. If a user likes both Metallica and Mozart, it does not necessarily mean that he wants to listen to them in the same session.

Another approach is to move the focus from the user-edited features to descriptors that can be automatically extracted from the music. These types of features should be more reliable as they require no intervention by a set of users. A recent survey by Aucouturier and Pachet [AP03] gives a comprehensive list of features used for music information extraction, although research in feature-extraction continues.

Once the features are extracted, a *machine learning* algorithm can be used to

classify into categories that match the user's taste. Machine learning is a field of computer science that, roughly speaking, analyzes the data using statistical algorithms. Such algorithms could, for instance, create a model that discriminates the categories using a labeled dataset as training data. This strategy has proven very successful, and has already been used in many products, such as MusicMagic, from Predixis (`predixis.biz`), or Musipedia (`www.musipedia.org`)[1].

Our contribution in this thesis is the description of several effective feature extraction techniques and learning algorithms that can be used to face the challenges of automatic music classification.

In this work we are using part of the work previously published by the author for the ICMC [CEK05a] and ISMIR conferences [EC05, CEK05b], and of a recent submission to the Machine Learning Journal [BCE+05].

1.1 Brief Introduction to Machine Learning

Machine learning generally refers to the area of artificial intelligence that concerns making computer "learn" a specific pattern or task. Different learning problems are addressed by different families of algorithms. There are grouped basically in this simplified taxonomy:

- **Supervised learning algorithms** generate a function that maps the inputs to desired outputs. Fo instance, in classification they are used in cases where each example has a set of attributes and a label, and the algorithm has to find a generalization of the attributes that characterize each label.

- **Unsupervised learning algorithms**, on the other hand, work in situations where ouputs (or labels in classification) are not provided for training. A common task of unsupervised learning is the automatic selection of "clusters" in the data in feature hyperspace, or in a transformation of it, such that missing attributes can be inferred.

[1] A list of music information retrieval systems is available at `http://mirsystems.info/index.php?id=mirsystems`.

- **Reinforcement learning algorithms**, learn a policy by interacting with the world. Each action has some impact on the environment, which returns a feedback (a reward or a penalty) that guides the learning algorithm to refine the actions it takes in various situations.

The learning algorithms used for this work are all supervised algorithms. These can be further subdivided into two types: regression and classification algorithms. In regression, the algorithm maps an observation onto a real value. In classification the chosen labels form a closed set, and the classifier maps an observation to a label that belongs to the set. Classification algorithms are the type of learner that interests us, as we want to classify sound into discrete labels that follows a specific, though not absolute, taxonomy.

To train supervised learning algorithms, we need a training set that contains n examples with the corresponding labels. Formally, training set D_n is defined as $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$[2] for binary problems. Each data point consists of an observation (or attributes) vector $\mathbf{x}_i = \left(x^{(1)}, \ldots, x^{(d)}\right) \in \mathbb{R}^d$ containing the description of the example, and a binary label $y_i \in \{-1, 1\}$ that tells in which class the example belongs to.

Each example can be considered as a point in the hyperspace of features; that is, each component of the observation vector is a value that defines the coordinate inside the hyperspace. For instance if the dataset represents pets, the feature set might be

$$Pet : \{Height, Width, Weight, Legs, Furryness\},$$

and each point defined as a coordinate in this six dimensions space, for instance an example belonging to class *cat* might have this observations

$$\mathbf{x}_{cat} = \{20, 40, 4.3, 4, 1.1\}.$$

[2]Throughout this work we will use uppercase symbols (e.g., A) to denote real-valued matrices, bold symbols (e.g., $\mathbf{x}$) for real-valued vectors, and their italic counterparts (e.g., x_i or $x^{(i)}$) to refer to elements of these vectors.

In the binary case, our examples will be labeled either "cat" (1) or "other" (-1).

The job of the learning algorithm is to find the parameters within its own model (and which depend on the type of algorithm) by analyzing the training set D_n. The resulting function $f(\mathbf{x})$ will use these parameters to return a binary decision on example $\mathbf{x}$.

The classifier is assumed to be able to predict the correct output for examples, not present in the original training set, thus to *generalize.* A natural way to evaluate the error of the classifier is by looking at the misclassification rate, that is

$$\hat{R} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\{f(\mathbf{x}_i) \neq y_i\}$$

where the indicator function $\mathbb{I}\{A\}$ is 1 if its argument A is true and 0 otherwise.

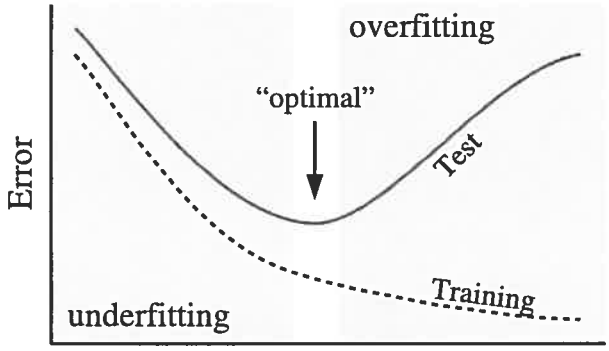


Figure 1.1: An example of training process, in which the training error $\hat{R}_{train}$ reaches zero after some time, as the algorithm adjust itself to match very specific features of the training data. As result, the error on a held-out test set increase.

During learning, the algorithm evaluates the examples of the training set, and as it goes on, it generally has the training error $\hat{R}_{train}$ going down. However, the learning process might become too specific. For instance if the algorithm adjust its parameters to match very specific features of the training data, like the values of $20, 21, 24$ for the height of cats, it will perfectly classify all the cats with these specific characteristics, but not the ones with height 23. This situation is defined

as *overfitting*, and is depicted in Figure 1.2.

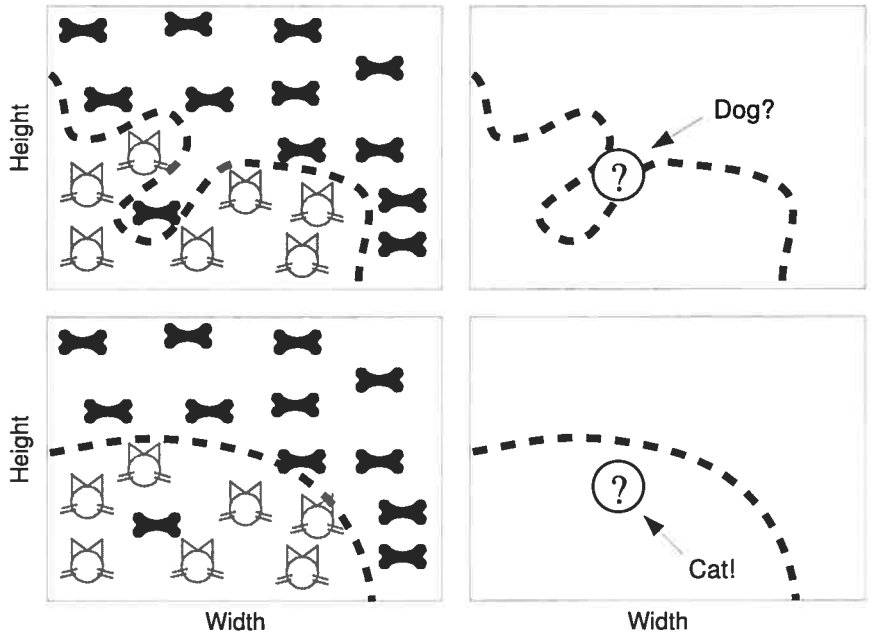


Figure 1.2: Cat and dogs could be classified just considering their size. The vast majority of cats are smaller than the vast majority of dogs; however, some dogs might be particularly small. A learning algorithm that overfits would draw an extremely complicate decision boundary to include the single case in which the dog is very small (upper left). This boundary will fail to classify correctly an animal, depicted by a question mark point, that is very likely to be a cat. A learning algorithm *generalizes* in the sense that it sticks more with general trends, rather than specific cases, as in the bottom figures.

An algorithm must be able to obtain low error on a held-out dataset, called *test set*, for which the error $\hat{R}_{test}$ decline as the learning process progresses, and raise when the algorithm start overfitting. The learning algorithm must be stopped when the model returns the lowest test error, as showed in Figure 1.1. If this is done before the minimum has been reached, the algorithm is *underfitting*. A learning algorithm is capable of *generalization* if it can balance between overfitting and underfitting.

Often, to avoid a bias toward the test set (that is, having chosen the perfect parameter for the test set, which might not correspond to the optimal one in a general sense), a third set, called *validation* is used as stopping criterion, and the test set for evaluation.

AdaBoost algorithms, which we will discuss in Chapter 2 have the interesting property of being very resistant to overfitting in many real-world problems, with a test curve that keeps decreasing even after the training error reaches zero.

1.2 Structure of the Thesis

We propose to use AdaBoost as a learning algorithm to create the models for our classification tasks. Introduced by Freund and Schapire in 1997 [FS97], this algorithm is one of the best general purpose learning methods developed in the last decade. It has inspired several learning theoretical results and, due to its simplicity, flexibility, and excellent performance on real-world data, it has gained popularity among practitioners. Chapter 2 is mainly dedicated to describing this algorithm.

In Section 2.1 we briefly go through history of this algorithm, and in the following sections we discuss the important aspects that characterize it, its extensions, and how to implement it. In particular, in Section 2.5 we show how to extend AdaBoost in order to solve multi-class problems, that is, problems with more than two labels.

This algorithm is particularly interesting because it has proven effective in dealing with the large number features that we are using. A single song can have hundreds of feature elements. We call these elements *dimensions* because a single observation could be described as a point in a hyperspace where each element is a dimension. In such a huge space it is very difficult to regroup points as they "look" very sparse, unless the number of examples is very large. This is known as the *curse of dimensionality* [Bel61]. With our experiments in Section 4.3 we show how AdaBoost can be very competitive on these problems, as it selects and

aggregates only the elements of the features that are needed for the classification.

Music has a complex structure that warrants several parallel approaches. Using the simple raw signal alone is not a viable approach, for reasons related to psychoacoustic and computational power that we discuss in Chapter 3. Instead, we are relying on features that capture various aspects of music that are important for humans. This work is motivated by knowledge from signal processing theory, physics of sound, psychoacoustics, human auditory perception and music theory.

Li and Tzanetakis [LT03b] define three types of auditory features: the timbral texture, the rhythm, and the pitch. Timbre is the sound-quality of near-instants, or simply the quality of a musical sound. We present several standard techniques to extract this kind of features in Section 3.1, and in Section 3.2 we show how we can blend their output together to cut down the size of the generated data, without reducing the quality of the information.

Rhythm involves the variation and duration of patterns of sounds in time. The structures that are defined by these patterns are ultimately the essence of music. In Section 3.3 we discuss two different approaches to capture one basic component of rhythm: musical tempo.

Finally, pitch is the psychoacoustic perception of the frequency of a sound (i.e. a note). We do not discuss pitch here because it is extremely hard to capture in polyphonic sounds, and current methods are fairly unreliable.

The last part of the chapter, Section 3.4, is dedicated to a novel approach that uses a robust image classifier to learn patterns in a visual representation of music. The basic idea is that if music is rendered as an image, for instance using a short discrete Fourier transformation (SDFT) that plots frequency on one axis and time on another, we can use simple visual features – called Haar-like features – to map the "areas" that characterize the sound we are trying to learn. We proved the effectiveness of this technique by building a robust speech/music discriminator, discussed in Section 4.2.

In Chapter 4 we will show the results of experiments on the algorithm and the features. In Section 4.1 we will evaluate the hyper-parameters of AdaBoost using

a well known database as testbed.

We use timbral and rhythmical features and AdaBoost to build a state-of-the-art classifier for music genre, described in Section 4.3, and in the following section we show the results of MIREX 2005 international contest in music information extraction, where our audio genre recognition algorithm ranked first in audio genre classification among 15 submissions.

CHAPTER 2

CLASSIFIERS

In this chapter we will primarily discuss boosting algorithms, and in particular AdaBoost, which has proved in our experiments to be very effective for solving problems related to information music retrieval.

For genre classification (see Section 4.3) we compare AdaBoost with Support Vector Machines (SVMs) and Artificial Neural Networks (ANNs), therefore we will discuss them briefly in Section 2.6. For a more complete treatment of SVMs see [HTF01]. For a more complete treatment of ANNs see [Bis95].

2.1 Boosting

There is an old saying that states "there is strength in numbers", that is, the result of a group can be higher than the simple sum of its parts. This is also, in some extent, true for machine learning.

Research in so-called *ensemble* learning algorithms has been made since long time. The simplest method that uses this idea is similar to democracy: we have a set of *experts* that give an opinion on an input example. The final vote is an average over these opinions. We define our expert as a function $h(\mathbf{x})$ that receives an input $\mathbf{x}$ and returns a positive or negative vote: formally $\{h : \mathbb{R}^d \to \{-1, +1\}\}$. The averaged learner $f(\mathbf{x})$ is the result of the linear combination of the functions

$$f(\mathbf{x}) = \frac{1}{N} \sum_j h^{(j)}(\mathbf{x}),$$

for N functions.

This is generally defined as *ensemble of experts*, and it is often applied when the experts come from very different areas, which in machine learning can mean that the algorithms are very different. It is important to note that in the previous

equation the experts evaluate the same attributes of the same example. This might not always be true, as there may be experts that treat different aspects of a problem. These aspects must be correlated in some sense, or else the experts will end up forming opinions based on incompatible evidence.

Ensembles of experts are similar to the strategy of another popular algorithm called *bagging*. With bagging, also known as *bootstrap aggregation*, we generate J subsets $\{D_1, \cdots, D_j\}$ of data by sampling from a training set D. We then learn a model on each subset, and finally combine the models by averaging the output (in the case of regression), or voting (in the case of classification).

This method is often used with a single algorithm repeated many times, instead of a number of different algorithms, and it reduces variance and generally helps to avoid over-fitting. The subsets D_j can also have the same size of D, and in this case all examples will be repeated in each D_j.

Schapire [Sch90] did a further development of this idea, defined as **boosting**, which was the first simple procedure in the PAC-learning framework theorized in 1984 by Valiant [Val84]. In Valiant's model the notion of successful learning is formally defined using probability theory. A learner is said to have *accuracy* $(1-\epsilon)$ with *reliability* $(1-\delta)$ if it generates a binary hypothesis whose accuracy is at least $(1-\epsilon)$ with probability at least $(1-\delta)$, where ϵ is the error of the learner, and δ is a parameter which defines the confidence. The probability that the algorithm fails is measured with respect to a random choice of examples given in the learning algorithm and in a possible internal randomization of the algorithm. Very roughly speaking, the learner gets samples that are classified according to a function from a certain class. The aim of the learner is to find an arbitrary approximation of the function with high probability.

Schapire showed that a learner, even if rough and moderately inaccurate, could always improve its performance by training two additional classifiers on filtered version of the input data stream. These algorithms, are called *weak learners* because their performance is guaranteed (with high probability) to be only slightly better than chance. After learning an initial classifier $h^{(1)}$ on the first n training

points,

- $h^{(2)}$ is learned on a new sample of n points, half of which are misclassified by $h^{(1)}$,
- $h^{(3)}$ is learned on n points from which $h^{(1)}$ and $h^{(2)}$ disagree,
- the boosted classifier is $h^{(B)} = \text{MAJORITYVOTE}(h^{(1)}, h^{(2)}, h^{(3)})$.

Schapire's "Strength of Weak Learnability" theorem proves that $h^{(B)}$ has improved performance over $h^{(1)}$.

With a number of learners larger than three, the algorithm is defined recursively. Each level of the recursion is a learner whose performance is better than the performance of the previous recursion level. The final hypothesis generated can be represented as a circuit consisting of many three-input majority gates.

In 1995 Freund [Fre95] proposed a "Boost by Majority" variation which combined many weak learners simultaneously in a *single layer* circuit, and improved the performance of the simple boosting algorithm of Schapire. The improvement is achieved by combining a large number of hypotheses, each of which is generated by training the given learning algorithm on a different set of examples.

The biggest weakness of this algorithm is the assumption that each hypothesis has a fixed error rate, and is therefore difficult to use in practice.

2.2 AdaBoost

Introduced in 1997 by Freund and Schapire [FS97], the ADABOOST algorithm solved many of the practical difficulties of the previous boosting algorithms. Unlike the previous boosting algorithms, ADABOOST needs no prior knowledge of the accuracies of the weak hypotheses. Rather, it adapts to these accuracies and generates a confidence parameter α that changes at each iteration according to the error of the weak hypothesis. This is the basis of its name: "Ada" is short for "adaptive".

For this section we assume the binary case. Therefore we have an input training set $D_n = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ where each label $y_i \in \{-1, +1\}$. The algorithm mantains a weight distribution $\mathbf{w}^{(t)} = (w_1^{(t)}, \ldots, w_n^{(t)})$ over the data points. In general, the weight w_i will express how hard it is to classify $\mathbf{x}_i$.

The basic AdaBoost algorithm is remarkably simple. In a series of rounds or iterations $t = 1, ..., T$ a weak learner $\text{Weak}(D_n, \mathbf{w})$ finds a binary weak hypothesis (or base classifier) h_t, coming from a subset $\mathcal{H}$ of $\{h : \mathbb{R}^d \to \{-1, +1\}\}$ appropriate for the distribution of weights $\mathbf{w}^{(t)}$, that is with the lowest weighted error. When this is not possible, a subset of the training examples can be sampled according to $\mathbf{w}^{(t)}$ (for instance with a stratified sampling of the weights which takes the hardest examples with higher probability), and these (unweighted) resampled examples can be used to train the weak learner.

At the beginning, the weight vector is initialized to be equal for each point. On each boosting round, the weight of incorrect examples is increased so that the weak learner is forced to focus on the hard examples on the training set. Figure 2.1 shows the evolution of this distribution on a simple two dimensional problem and a Decision Stumps weak learner (which we will see in Section 2.3).

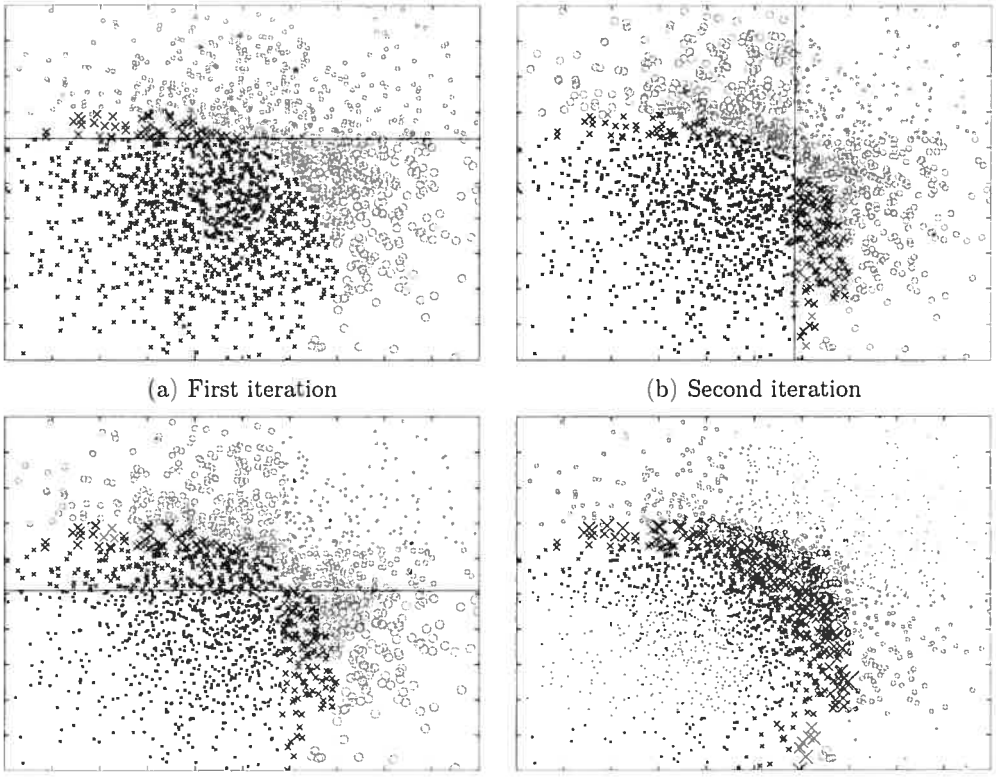
Formally the goal of the base classifier is to minimize the weighted error at iteration t

$$\epsilon^{(t)} = \epsilon^{(t)}(h) \triangleq \sum_{i=1}^{n} w_i^{(t)} \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) \neq y_i\right\}, \tag{2.1}$$

This is equivalent to maximizing the *edge*

$$\gamma^{(t)} \triangleq 1 - 2\epsilon^{(t)} = \sum_{i=1}^{n} w_i^{(t)} h^{(t)}(\mathbf{x}_i) y_i.$$

Freund and Schapire proved that for the binary case the error of the final hypothesis (with respect to the observations) is bounded by $\exp\left(-2\sum_{t=1}^{T} \gamma^{(t)2}\right)$. Since a weak hypothesis that makes an entirely random guess has error $1/2$, $\gamma^{(t)}$ measures the accuracy of the tth weak hypothesis relative to random guessing. The important consequence is that if we can consistently find weak hypotheses that are slightly



(a) First iteration (b) Second iteration

(c) Third iteration (d) 100th iteration

Figure 2.1: AdaBoost mantains a distribution of weights $\mathbf{w}$ on the examples, as showed by the size of the points. The four images represent the state of the weights after the first, second, third and 100th iteration. After 100 iterations it is clear how the weight focuses the attention of the weak learner on the hard examples near the boundary of the two classes. The first three figures show the threshold position of the weak learner used: a decision stump described in section 2.3.1.

better than coin flip, the training error of the final boosted classifier will drop *exponentially fast.*

A significant difference with the previous boosting algorithm is that the bound on the accuracy of the final hypothesis improves when *any* of the weak hypotheses is improved, where previously the performance bound depended only on the accuracy of the last accurate weak hypothesis. On the other hand if the weak hypotheses have the same accuracy, the performance of AdaBoost is very close to that achieved by the best of the preceding boosting algorithms.

Once the weak hypothesis $h^{(t)}$ has been received, AdaBoost chooses a coefficient $\alpha^{(t)}$ that measures the *confidence* assigned to $h^{(t)}$, and it is directly related to its error

$$\alpha^{(t)} = \frac{1}{2}\ln\left(\frac{1+\gamma^{(t)}}{1-\gamma^{(t)}}\right) = \frac{1}{2}\ln\left(\frac{1-\epsilon^{(t)}}{\epsilon^{(t)}}\right). \tag{2.2}$$

Note that $\alpha^{(t)} \geq 0$ because $\epsilon^{(t)} \leq 1/2$ (which we can assume without loss of generality), and that $\alpha^{(t)}$ gets larger as $\epsilon^{(t)}$ gets smaller.

The weight distribution is updated according to the following equation

$$\begin{aligned} w_i^{(t+1)} &= \frac{w_i^{(t)}}{Z^{(t)}} \times \begin{cases} \exp(-\alpha^{(t)}) & \text{if } h^{(t)}(\mathbf{x}_i) = y_i \text{ (correct prediction)} \\ \exp(\alpha^{(t)}) & \text{if } h^{(t)}(\mathbf{x}_i) \neq y_i \text{ (wrong prediction)} \end{cases} \\ &= \frac{w_i^{(t)} \exp\left(-\alpha^{(t)} h^{(t)}(\mathbf{x}_i) y_i\right)}{Z^{(t)}}, \end{aligned} \tag{2.3}$$

where

$$Z^{(t)} = \sum_{j=1}^{n} \left(w_j^{(t)} \exp\left(-\alpha^{(t)} h^{(t)}(\mathbf{x}_j) y_j\right)\right) \tag{2.4}$$

is a normalization factor chosen so that $\mathbf{w}^{(t+1)}$ will be a distribution, which means that

$$\sum_{i=1}^{n} w_i^{(t)} = 1. \tag{2.5}$$

Because the binary weak hypothesis has the form $h_t : \mathcal{X} \to \{-1, +1\}$ it is possible

to further simplify the update formula

$$w_i^{(t+1)} = \begin{cases} \frac{w_i^{(t)}}{2(1-\epsilon^{(t)})} & \text{if } h^{(t)}(\mathbf{x}_i) = y_i, \\ \frac{w_i^{(t)}}{2\epsilon^{(t)}} & \text{if } h^{(t)}(\mathbf{x}_i) \neq y_i, \end{cases}$$

and the normalization factor

$$Z^{(t)} = 2\sqrt{\epsilon^{(t)}(1-\epsilon^{(t)})}. \tag{2.6}$$

Finally the algorithm outputs the strong learner that includes, in a linear combination, a weighted average of the weak hypotheses

$$f^{(T)}(\cdot) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot), \tag{2.7}$$

and uses the sign of $f^{(T)}(\mathbf{x})$ to classify $\mathbf{x}$.

The algorithm must terminate if $\alpha^{(t)} \leq 0$ which is equivalent to $\gamma^{(t)} \leq 0$ and $\epsilon^{(t)} \geq 1/2$. If the base classifier set $\mathcal{H}$ is closed under multiplication by -1, $\alpha^{(t)}$ can be chosen non-negative, and $\alpha^{(t)}$ can be zero[1] only if all base classifiers in $\mathcal{H}$ have base errors $1/2$.

Algorithm 1 shows the pseudocode of AdaBoost algorithm. A real world implementation requires very little modification of this code.

For the analysis of the algorithm, we first define the *unnormalized margin* achieved by $f^{(T)}$ on $(\mathbf{x}_i, y_i)$ as

$$\rho_i \triangleq \rho_{f^{(T)}}(\mathbf{x}_i, y_i) = f^{(T)}(\mathbf{x}_i) y_i, \tag{2.8}$$

[1]Strictly speaking, $\alpha^{(t)} = 0$ could be allowed but in this case it would remain 0 forever so it makes no sense to continue.

Algorithm 1 Basic AdaBoost

Input: D_n, $\textsc{Weak}(D_n, \mathbf{w}^{(t)})$, T

$\mathbf{w}^{(1)} \leftarrow (1/n, ..., 1/n)$
for $t \leftarrow 1$ to T **do**
  $h^{(t)} \leftarrow \textsc{Weak}(D_n, \mathbf{w}^{(t)})$ {Find optimal weak classifier}
  $\epsilon^{(t)} \leftarrow \sum_{h^{(t)} \neq y_i} w_i^{(t)}$ {Compute the weighted error}
  if $\epsilon^{(t)} \geq 1/2$ **then**
    return $f^{(t-1)}(\cdot) = \sum_{j=1}^{t-1} \alpha^{(j)} h^{(j)}(\cdot)$
  end if
  $\alpha^{(t)} \leftarrow \frac{1}{2} \ln\left(\frac{1-\epsilon^{(t)}}{\epsilon^{(t)}}\right)$ {Compute the confidence}
  for $i \leftarrow 1$ to n **do**
    if $h^{(t)}(\mathbf{x}_i) \neq y_i$ **then**
      $w_i^{(t+1)} \leftarrow \frac{w_i^{(t)}}{2\epsilon^{(t)}}$ {Re-weight incorrectly classified points}
    else
      $w_i^{(t+1)} \leftarrow \frac{w_i^{(t)}}{2(1-\epsilon^{(t)})}$ {Re-weight correctly classified points}
    end if
  end for
end for
return $f^{(T)}(\cdot) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot)$ {Combined classifier}

and the (normalized) margin achieved by the normalized classifier $\tilde{f}^{(T)}$ as

$$\tilde{\rho}_i \triangleq \tilde{f}^{(T)}(\cdot) y_i \triangleq \frac{\rho_i}{\|\alpha\|_1} = \frac{\sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot)}{\sum_{t=1}^{T} \alpha^{(t)}}. \tag{2.9}$$

The *misclassification rate* can be seen as a function of the margin, that is,

$$\hat{R}(f^{(T)}) \triangleq \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\left\{\rho_i^{(T)} < 0\right\}. \tag{2.10}$$

Informally, one can think of the normalized margin of a training example as the distance (by some measure) from the example to the decision boundary $\{\mathbf{x} : \tilde{f}^{(T)}(\mathbf{x}) = 0\}$ and therefore the confidence in the classification of example $\mathbf{x}$. If $\rho_i < 0$ the example has been misclassified. If ρ_i is positive the confidence in the classification of $\mathbf{x}_i$ increases proportionally to $\rho_i > 0$.

Schapire et al. [SFBL97] suggested that the effectiveness of AdaBoost and

other weight-based algorithms, is due to their tendency to produce *large margin classifiers*. Roughly speaking, if a combination of learners correctly classifies most of the training data with a large margin, then its error is probably small.

In general, margin-based classifiers attempt to minimize a *margin loss function*

$$\hat{R}_L(f) \triangleq \frac{1}{n}\sum_{i=1}^{n} L\left(f(\mathbf{x}_i)y_i\right)$$

where $\hat{R}_L(f)$ is the misclassification rate expressed as empirical risk and L is an arbitrary loss function. In the case of Support Vector Machines (see Section 2.6.1), a popular margin-based algorithm, $L(\rho) = (1-\rho)_+$ or also $L(\rho) = (1-\rho)_+^2$, where the $(x)_+$ operator is defined as

$$(x)_+ = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases}$$

The simpler Regularized Least Square Classificators (RLSC) (but also neural networks) defines a loss function $L(\rho) = (1-\rho)^2 = (y_i - f(x_i))^2$. AdaBoost's loss function is exponential, that is

$$L(\rho) = e^{-\rho}. \tag{2.11}$$

which means that the exponential risk is

$$\hat{R}_e(f^{(T)}) = \frac{1}{n}\sum_{i=1}^{n} \exp(-f^{(T)}(\mathbf{x}_i)y_i) = \frac{1}{n}\sum_{i=1}^{n} \exp(-\rho_i^{(T)}). \tag{2.12}$$

Notice that since $e^{-x} \geq \mathbb{I}\{x \leq 0\}$ we have

$$\hat{R}(f^{(T)}) \leq \hat{R}_e(f^{(T)}), \tag{2.13}$$

that is, the exponential risk is an upper bound of the training error.

By repeated application of the weight update formula (2.3), we get

$$\begin{aligned} w_i^{(t+1)} &= w_i^{(1)} \prod_{j=1}^{t} \frac{\exp\left(-\alpha^{(j)} h^{(j)}(\mathbf{x}_i) y_i\right)}{Z^{(j)}} \\ &= \frac{1}{n} \frac{\exp\left(-\sum_{j=1}^{t} \alpha^{(j)} h^{(j)}(\mathbf{x}_i) y_i\right)}{\prod_{j=1}^{t} Z^{(j)}} \\ &= \frac{\exp\left(-\rho_i^{(t)}\right)}{n \prod_{j=1}^{t} Z^{(j)}}. \end{aligned} \tag{2.14}$$

Now it is possible to demonstrate the previous claim that the training error will decrease exponentially fast:

$$\hat{R}(f^{(T)}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\left\{\rho_i^{(T)} < 0\right\} \tag{2.15}$$

$$\leq \frac{1}{n} \sum_{i=1}^{n} \exp(-\rho_i^{(T)}) \tag{2.16}$$

$$= \sum_{i=1}^{n} w_i^{(t+1)} \prod_{t=1}^{T} Z^{(t)} \tag{2.17}$$

$$= \prod_{t=1}^{T} Z^{(t)} \tag{2.18}$$

In (2.15) we used the definition (2.10). For the inequality (2.16) we used (2.13), in (2.17) we used (2.14), and finally in (2.18) we used (2.5).

If we assume that our weak hypothesis $h^{(t)}$ returns an error $\epsilon^{(t)} \leq 1/2 - \delta$ slightly better than chance, it is easy to show that the training error is bounded[2] by $\hat{R}_e(f) \leq e^{-2T\delta^2}$ and becomes zero after at most

$$T = \left\lceil \frac{\ln n}{2\delta^2} \right\rceil + 1$$

iterations.

[2]A detailed description with all the passages is given at page 15 of [FS97].

The upper bound of (2.18) and (2.12) suggest that if the goal is to minimize the training error $\hat{R}(f^{(T)})$ (through the exponential margin cost $\frac{1}{n}\sum_{i=1}^{n} e^{-\rho_i}$) in a greedy fashion, then we should minimize $Z^{(t)}$ in each iteration. Indeed, for any given $\{-1,+1\}$-valued weak classifier $h^{(t)}$, its coefficient $\alpha^{(t)}$ is chosen by

$$\begin{aligned} \alpha^{(t)} &= \arg\min_{\alpha} Z^{(t)} \\ &= \frac{1}{2}\ln\left(\frac{1+\gamma^{(t)}}{1-\gamma^{(t)}}\right) = \frac{1}{2}\ln\frac{1-\epsilon^{(t)}}{\epsilon^{(t)}}, \end{aligned} \tag{2.19}$$

which explains (2.2). Now the choice of $h^{(t)}$ is based on

$$\begin{aligned} h^{(t)} &= \arg\min_{h:\alpha\geq 0} Z^{(t)} \\ &= \arg\min_{h:\epsilon^{(t)}\leq 1/2}\left(\epsilon^{(t)}\sqrt{\frac{1-\epsilon^{(t)}}{\epsilon^{(t)}}} + (1-\epsilon^{(t)})\sqrt{\frac{\epsilon^{(t)}}{1-\epsilon^{(t)}}}\right) \\ &= \arg\min_{h:\epsilon^{(t)}\leq 1/2} 2\sqrt{\epsilon^{(t)}(1-\epsilon^{(t)})} = \arg\min_{h}\epsilon^{(t)}, \end{aligned} \tag{2.20}$$

which explains the goal of minimizing the weighted error (2.1). In the case when the minimum alpha (2.19) cannot be obtained analytically (for example, real-valued classifiers), we can minimize $Z^{(t)}$ using line search. The procedure is simple since $Z^{(t)}$ *is convex in* α, however, it may be too time consuming to compute it for every weak classifier $h \in \mathcal{H}$ (or even impossible if $|\mathcal{H}|$ is infinite). In this case we can follow Mason et al.'s [MBBF00] functional gradient descent approach, in which at each iteration AdaBoost chooses a direction in which $Z^{(t)}$ is decreasing the fastest at $\alpha = 0$, that is

$$\begin{aligned} h^{(t)} &= \arg\max_{t} \left.\frac{\partial Z^{(t)}}{\partial\alpha}\right|_{\alpha=0} \\ &= \arg\max_{t}\sum_{i=1}^{n} w_i^{(t)} h(\mathbf{x}_i) y_i = \arg\max_{h}\gamma^{(t)}, \end{aligned} \tag{2.21}$$

and then find the optimal coefficient only for the selected base classifier. In

the case of $\{-1, +1\}$-valued weak learners the two optimizations (2.20) and (2.21) coincide, which means that (2.21) yields the optimal $h^{(t)}$ that also minimizes $Z^{(t)}$. In other cases (2.21) is suboptimal but the convergence is guaranteed (but slower) as long as the edge $\gamma^{(t)} > 0$ at each iteration.

2.3 Weak Learners

As previously said, AdaBoost needs a base function $h(\mathbf{x})$ which has the ability of classifying just better than chance. This is a big advantage, because weak learners can be chosen with little or no prior knowledge lending flexibility. It is however important to remember that if the base function is not *weak enough* the resulting strong learner might overfit easily. In fact boosting seems to be especially susceptible to noise [Die00] in such cases. In this section we discuss two different basic types of binary weak learners. In Section 2.5.2 the same weak learners will be extended to deal with multi-class problems.

2.3.1 Decision Stumps

As we previously said, the goal of the base classifier is to minimize the weighted error at each iteration t. If there is no a-priori knowledge available on the domain of the learning problem, we choose simple classifiers, such as *decision trees* or, the even simpler decision stumps (decision trees with two leaves). In this work we are primarily using decision stumps as weak learners. In the next section we discuss a variation of the typical tree algorithm that uses previously found stumps to create its structure. Therefore we are introducing decision stumps here first.

A decision stump can be defined by two parameters, the index j of the attribute that it cuts and the threshold q of the cut. Formally,

$$\varphi_{j,q}(\mathbf{x}) = 2\mathbb{I}\left\{x^{(j)} \geq q\right\} - 1 = \begin{cases} 1 & \text{if } x^{(j)} \geq q, \\ -1 & \text{otherwise,} \end{cases}$$

where $x^{(j)}$ is the jth element of the feature vector $\mathbf{x}$. To obtain a "symmetric" class of base classifiers $\varphi_{j,q}(\mathbf{x})$ we multiply by a $\{-1,1\}$-valued "alignment" value v. Intuitively, $v=-1$ means that the weighted error on the training set using $\varphi_{j,q}$ is larger than 0.5. This means that the decision needs simply to be "flipped" in order to get an error smaller than coinflip. We use this notation because it can be easily extended to the multi-class case that we discuss in Section 2.5.

To minimize (2.1), in each iteration we exhaustively search over the features $j=1,\ldots,d$, thresholds q, and alignment values v to obtain

$$\left(j^{(t)},q^{(t)},v^{(t)}\right)=\arg\min_{j,q,v}\epsilon\left(\varphi_{j,q}(\cdot)v\right), \tag{2.22}$$

and define the optimal weak classifier as

$$h^{(t)}(\cdot)=\varphi_{j^{(t)},q^{(t)}}(\cdot)v^{(t)}.$$

If the number of features d is small enough, it is possible to order the vector projections $\left(x_1^{(j)},\ldots,x_n^{(j)}\right)$, and store them in memory. In this case, considering that the alignment value v can be optimized element-wise, the cost of the weak learning of both strategies is $O(nd)$, so the whole algorithm runs in $O\left(nd(T+\log n)\right)$ time.

In Chapter 3 we describe a type of feature that has a large number of configurations, most of which are not explored due to random sampling. Hence, the vector projection of the feature must be sorted at each iteration, even if this is not necessary for all the possible configurations.

As we have seen, decision stump weak learners do have to go through all the features j to find the weak hypothesis that minimizes the weighted error. In our setup, only the feature that returns the lowest error with the given threshold is selected, but this is not a limitation imposed by AdaBoost, and the weak learner could well be a grouping of the best m features, or a combination of them.

One interesting consequence of forcing the selection of one single dimension every iteration, is that we automatically obtain a feature selection algorithm, as

done by Tieu and Viola [TV01]. Moreover, it has been suggested that if the selected features are forced not to be selected again, AdaBoost becomes a supervised dimensionality reduction algorithm [RA04].

2.3.2 Decision Trees

A Decision Stump is the main type of weak learner used in this work, but has the disadvantage that the classification is always linear in the space of the stumps outputs, even within the AdaBoost framework. The problem is that certain data cannot be separated by a linear combination of stumps, and therefore the learner underfits. A simple example of this type of problem is XOR (see [DHS01] page 285) depicted in Figure 2.2. XOR can only be solved by a tree having at least two levels. (Alternatively *kernel trick* [ABR64] that is a transformation to a higher dimensional representation of the data, can be used to change the topology of the space in a way that allows linear separation. But in this case any linear separator would work.)

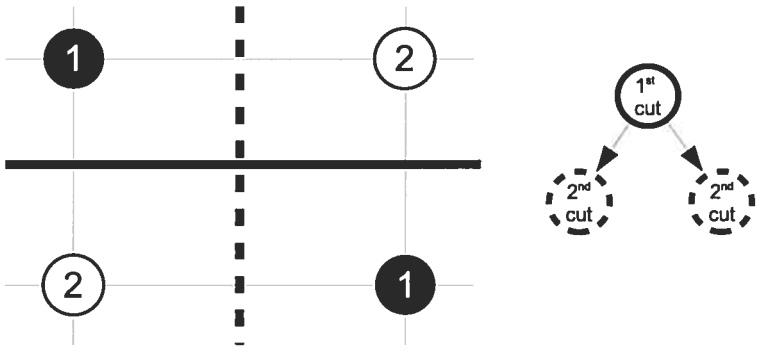


Figure 2.2: The XOR problem cannot be solved by a decision stumps or a combination of them. However, by using a simple two-level tree it is possible to partition the feature's space and obtain a perfect solution.

Most real-life problems, however, are solved by stumps with performance similar to more sophisticated weak learners, even if stumps tend to converge at a slower rate, especially if d is large. Haar-like features, the procedural type of features that

we describe in Section 3.4 operates as a sort of kernel-trick and can overcome the limitations of stumps.

We implemented a simple version of a decision tree in which each level defines a threshold over a feature as showed in Figure 2.3. The classification begins at the first level (on the top) and, depending on the response to a particular property of the pattern, proceeds through successive links until it reaches the terminal nodes, defined as *leaves*, where the class is returned.

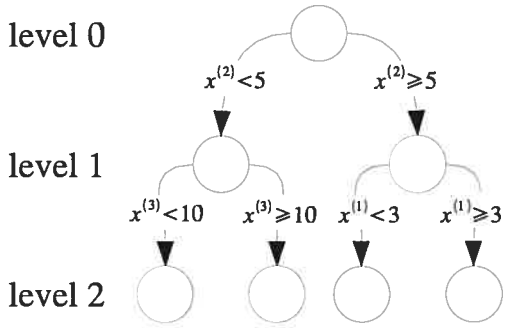


Figure 2.3: A possible configuration of a decision tree with two levels.

Trees can help solve problems that are beyond the reach of stumps, but because for each node we need to find the optimal values of formula (2.22), this approach might be too costly for large databases, or for values of d particularly big.

A solution to this problem is a "constructive" approach. At each boosting iteration t we find the optimal weak classifier $h^{(t)}$ as usual, but if $t > 2$, we also build a tree using just the features $j^{(1)}, \cdots, j^{(t-1)}$ found in previous iterations. If the error of the tree is lower than the one found with stumps, then it is returned as the weak hypothesis for the current iteration.

In the case of a large feature space, that is with $d > T$, this approach can drastically reduce the number of features evaluated. With procedural features (see Section 3.4), having a huge number of possible configurations, this is a compelling simplification, as going through all configurations would be too costly. This method can be also used in the case of features that are particularly expensive to compute,

as their values can be stored at each iteration for later use.

2.4 AdaBoost Extensions

It is possible to extend the basic AdaBoost algorithm to avoid some of its weaknesses and obtain better performance. In this section we review two major extensions that have been used in this work.

2.4.1 Abstention

As we have seen, in the typical AdaBoost the binary weak learner $h^{(t)} : \mathbb{R} \rightarrow \{-1, +1\}$, and is therefore forced to give an "opinion" for each example $\mathbf{x}_i$. This is not always desirable as the weak learner might have not be suited to classify every $\mathbf{x} \in \mathbb{R}$. This is often the case when among the set $\mathcal{H}$ of weak hypotheses there are some that are specialized for a subset of the examples. In such case the overall performance of the final classifier $f^{(T)}$ would benefit from them, but only when they are used on their specialized subset of the input.

The solution to this problem is trivial as long as we have a weak learner that *knows when* to abstain. Formally, the abstention base classifier has the form $h^{(t)} : \mathbb{R}^d \rightarrow \{-1, 0, +1\}$, thus we can calculate Z from (2.4) as

$$\begin{aligned} Z^{(t)} &= \sum_i w_i^{(t)} \exp\left(-\alpha u_i^{(t)}\right) \\ &= \sum_{b \in \{-1,0,+1\}} \sum_{i:u_i^{(t)}=b} w_i \exp\left(-\alpha^{(t)} u_i^{(t)}\right) \\ &= \epsilon_0^{(t)} + \epsilon_-^{(t)} e^{\alpha^{(t)}} + \epsilon_+^{(t)} e^{-\alpha^{(t)}} \end{aligned}$$

where

$$u_i^{(t)} = h^{(t)}(\mathbf{x}_i) y_i,$$

and

$$\begin{aligned}
\epsilon_+^{(t)} &= \sum_{i=1}^{n} w_i^{(t)} \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) y_i = +1\right\}, \\
\epsilon_-^{(t)} &= \sum_{i=1}^{n} w_i^{(t)} \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) y_i = -1\right\}, \\
\epsilon_0^{(t)} &= \sum_{i=1}^{n} w_i^{(t)} \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) = 0\right\},
\end{aligned}$$

which means respectively the sum of weights for correctly classified, incorrectly classified, and abstained points. It is also important to remember that $\epsilon_+^{(t)} + \epsilon_-^{(t)} + \epsilon_0^{(t)} = 1$. The coefficient $\alpha^{(t)}$ is chosen by minimizing $Z^{(t)}$ as usual

$$\begin{aligned}
\alpha^{(t)} &= \arg\min_{\alpha} Z^{(t)} \\
&= \frac{1}{2} \ln \left(\frac{\epsilon_+^{(t)}}{\epsilon_-^{(t)}} \right).
\end{aligned}$$

This formula indicates that the confidence parameter is computed only by taking into consideration the examples in which our weak learner is able to give an opinion. With these settings the objective function to minimize (2.20) changes into

$$\begin{aligned}
h^{(t)} &= \arg\min_{h:\epsilon_+^{(t)} > \epsilon_-^{(t)}} Z^{(t)} \\
&= \arg\min_{h:\epsilon_+^{(t)} > \epsilon_-^{(t)}} \left(\epsilon_0^{(t)} + 2\sqrt{\epsilon_+^{(t)} \epsilon_-^{(t)}} \right)
\end{aligned}$$

In the case where the weak learner makes no mistakes (thus $\epsilon_-^{(t)} = 0$), we have to modify our way to compute $\alpha^{(t)}$ in order to avoid the zero at the denominator. There are two possible ways to solve this problem, the first one is simply to add a small value δ:

$$\alpha^{(t)} = \frac{1}{2} \ln \left(\frac{\epsilon_+^{(t)} + \delta}{\epsilon_-^{(t)} + \delta} \right).$$

The second solution is a little bit more sophisticated and involves the concept

of a margin threshold being bigger than zero. This additional extension will be discussed in the following section.

2.4.2 Regularization

Despite the greedy minimization of the training error, AdaBoost has proved to be very resistant to overfitting, and capable of decreasing the test error even if the training error has been driven to zero. There are however some cases in which this algorithm can overfit if it is run long enough. To avoid this problem, the algorithm is generally stopped early by validating the number of iterations T on a validation set.

An alternative to solve this problem is *regularization*, that is the introduction of an *edge offset* parameter $\theta \geq 0$ in the confidence formula (2.2)

$$\begin{aligned}
\alpha^{(t)} &= \arg\min_{\alpha} \left(Z^{(t)} e^{\theta\alpha}\right) \\
&= \frac{1}{2}\ln\left(\frac{1+\gamma^{(t)}}{1-\gamma^{(t)}} \times \frac{1-\theta}{1+\theta}\right) \\
&= \frac{1}{2}\ln\left(\frac{1+\gamma^{(t)}}{1-\gamma^{(t)}}\right) - \frac{1}{2}\ln\left(\frac{1-\theta}{1+\theta}\right) \\
&= \frac{1}{2}\ln\left(\frac{1-\epsilon^{(t)}}{\epsilon^{(t)}}\right) - \frac{1}{2}\ln\left(\frac{1+\theta}{1-\theta}\right).
\end{aligned}$$

This formula shows how the confidence α is decreased by a constant term every iteration, suggesting a mechanism similar to weight decay for reducing the effect of the overfitting. The algorithm must terminate if

$$\gamma^{(t)} \leq \theta \tag{2.23}$$

which means that none of the weak hypotheses in $\mathcal{H}$ has an edge equal or smaller than θ. For $\theta = 0$ finding hypotheses with the edge larger than 0 is not hard to achieve if $\mathcal{H}$ is closed under multiplication by -1.

If we assume the margin parameter θ to be fixed the original loss function (2.11)

becomes

$$L_e^{(\theta)} = e^{-\rho_i + \theta\|\alpha\|_1}$$

The marginal misclassification rate (2.10) becomes:

$$\hat{R}^{(\theta)}(f) = \sum_{i=1}^{n} \mathbb{I}\{\tilde{\rho}_i < \theta\}.$$

Minimizing $\hat{R}^{(\theta)}(f)$ with $\theta > 0$ has the effect of pushing the points away from the decision border in the feature space.

Following the same procedure we used to find (2.18), we get an upper bound of the training error

$$\hat{R}^{(\theta)}(f^{(T)}) \leq \hat{R}_e^{(\theta)}(f) = 2^T \prod_{t=1}^{T} \sqrt{\epsilon^{(t)1-\theta}(1-\epsilon^{(t)})^{1+\theta}}.$$

If we combine both regularized AdaBoost and abstention, our $Z^{(t)}$ becomes

$$Z^{(t)} = e^{\theta\alpha}\left(\epsilon_0^{(t)} + \epsilon_-^{(t)} e^{\alpha^{(t)}} + \epsilon_+^{(t)} e^{-\alpha^{(t)}}\right).$$

We can still compute α analytically:

$$\alpha^{(t)} = \begin{cases} \ln\left(-\frac{\theta\epsilon_0^{(t)}}{2(1+\theta)\epsilon_-^{(t)}} + \sqrt{\left(\frac{\theta\epsilon_0^{(t)}}{2(1+\theta)\epsilon_-^{(t)}}\right)^2 + \frac{(1-\theta)\epsilon_+^{(t)}}{(1+\theta)\epsilon_-^{(t)}}}\right) & \text{if } \epsilon_-^{(t)} > 0, \\ \ln\left(\frac{(1-\theta)\epsilon_+^{(t)}}{\theta\epsilon_0^{(t)}}\right) & \text{if } \epsilon_-^{(t)} = 0. \end{cases} \tag{2.24}$$

Thus by setting $\theta > 0$ we avoid the singularity if the error $\epsilon_-^{(t)}$ is equal to zero. The objective function h with the optimal α now becomes

$$\begin{aligned} h^{(t)} &= \underset{h:\epsilon_+^{(t)} > \epsilon_-^{(t)}}{\arg\min}\, Z^{(t)} \\ &= \underset{h:\epsilon_+^{(t)} > \epsilon_-^{(t)}}{\arg\min}\left(2\sqrt{\epsilon_+^{(t)}\epsilon_-^{(t)}} + \epsilon_0^{(t)}\right). \end{aligned} \tag{2.25}$$

The mixed algorithm is summarized in Algorithm 2.

Algorithm 2 AdaBoost with margin and abstention

Input: D_n, WEAK$(D_n, \mathbf{w}^{(t)})$, T, θ

$\mathbf{w}^{(1)} \leftarrow (1/n, ..., 1/n)$
for $t \leftarrow 1$ to T **do**
  $h^{(t)} \leftarrow$ WEAK$(D_n, \mathbf{w}^{(t)})$ {Find optimal weak classifier that minimize (2.25)}
  $\gamma_i^{(t)} \leftarrow w_i^{(t)} h^{(t)}(\mathbf{x}_i) y_i$ {Compute the edge}
  if $\sum_i \gamma_i^{(t)} \leq \theta$ **then**
    return $f_{t-1}(\cdot) = \sum_{j=1}^{t-1} \alpha^{(j)} h^{(j)}(\cdot)$
  end if
  $\alpha^{(t)} \leftarrow$ Equation (2.24) {Compute the confidence}
  for $i \leftarrow 1$ to n **do**
    $w_i^{(t+1)} = w_i^{(t)} \frac{\exp\left(-\alpha^{(t)} h^{(t)}(\mathbf{x}_i) y_i\right)}{Z^{(t)}}$ {Re-weight the points}
  end for
end for
return $f_T(\cdot) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot)$ {Combined classifier}

2.5 Multi-class AdaBoost

So far we have been concentrating on binary ADABOOST, in which we assign examples to one of two categories. This is a simple and well understood scenario, but it is not directly applicable to some of the problems that we are discussing in this work.

Intuitively, we define a multi-class problem as the classification of an example $\mathbf{x}_i$ which belongs to a (single) class from the set of classes. Formally, in the label vector $\mathbf{y}_i$ where

$$y_{i,\ell} = \begin{cases} 1 & \text{if } \mathbf{x}_i \text{ belongs to class } \ell \\ -1 & \text{otherwise,} \end{cases}$$

there is a single positive value for example $\mathbf{x}_i$.

There are several problems that are similar to multi-class classification, such as multi-label, where $\mathbf{y}_i$ contains more than one positive entries, or ranking in which the order of the labels is important. All of them can be understood as simpler

versions of a multi-variate regression

two-class $\subseteq$ multi-class $\subseteq$ multi-label $\subseteq$ ranking $\subseteq$ multi-variate regression.

In this work we will concentrate on multi-class problems, even if the extension to multi-label for two of the algorithms that we discuss (AdaBoost.MH and AdaBoost.MO) is trivial. This can be useful for example if the categories are organized in a hierarchical structure, so for instance a song can belong to more than one categories at the same time.

2.5.1 AdaBoost.M1

Probably the most simple and straightforward way to extend AdaBoost to deal with multi-class problems is AdaBoost.M1, proposed by Freund and Schapire in their original AdaBoost article [FS97]. In their approach the weak learner is able to generate hypotheses for each instance of the k possible labels, which means that the weak learner is a full multi-class algorithm itself. The AdaBoost algorithm does not need to be modified in any sense.

As in for the binary case, provided that each hypothesis has a prediction error less than a half (with respect to the set on which it was trained), the error decrease exponentially. However this requirement is much stronger than the binary case ($k = 2$), when the random guess is correct with probability $1/2$. In fact with $k > 2$ the probability is just $1/k < 1/2$. Thus, this method fails if the weak learner cannot achieve at least 50% accuracy *on all classes* when run on hard problems. This reflects the basic problem of this approach: a weak learner that concentrates its efforts on few classes is useless, because the poor performance on the other classes will hinder the final classification rate.

2.5.2 AdaBoost.MH

To address this weakness, several more sophisticated methods have been developed. These generally work by reducing the multiclass problem to a larger binary problem, with the so called *one-versus-all* approach.

Schapire and Singer's algorithm [SS98] **AdaBoost.MH** uses a slightly different strategy, which requires more elaborate communication between the boosting algorithm and the weak learner, but it has the advantage of giving the weak learner more flexibility in making its prediction. In ADABOOST.MH the weak learner receives a distribution of weights which is on the data *and* the classes, so that it considers each example $\mathbf{x}$ and each possible label ℓ in the form

For example x, is the correct label ℓ or is it one of the other labels?

This is important because as boosting progresses, training examples and their corresponding labels that are hard to predict, get incrementally higher weights while examples and labels that are easy to classify get lower weights. For instance, for the music genre classification problem, it might be easy to exclude "classical" from a piece of *The Beatles*, but hard to determine whether or not it belongs to "rock" or "pop". This forces the weak learner to concentrate on examples and labels that will make the most difference in an accurate classifier.

If no a-priori knowledge is given about the "similarities" of classes, and $\mathbf{x}_i$ can be potentially misclassified to any class, then the weight vector should be initialized to[3]

$$w_{i,\ell}^{(1)} = \begin{cases} \frac{1}{2n} & \text{if } \ell \text{ is the correct class (if } y_{i,\ell} = 1), \\ \frac{1}{2n(k-1)} & \text{otherwise (if } y_{i,\ell} = -1). \end{cases}$$

The weight distribution will be updated at the end of each iteration, and normalized so that for all $t = 1, \ldots, T$,

$$\sum_{i=1}^{n} \sum_{\ell=1}^{k} w_{i,\ell}^{(t)} = 1.$$

In general, the weight $w_{i,\ell}$ will express how "hard" it is to classify $\mathbf{x}_i$ into its correct class (if $y_{i,\ell} = 1$) or to differentiate it from the incorrect classes (if $y_{i,\ell} = -1$),

[3]This is based on speculation, but showed to be effective. The intuition behind this choice is that it will create well-balanced one-against-all problems in the beginning.

where the sum

$$\|\mathbf{w}_i\|_1 = \sum_{\ell=1}^{k} w_{i,\ell},$$

gives this measure on a data point $(\mathbf{x}_i, \mathbf{y}_i)$, similarly to what happen with the binary classification.

The weak learner now has to take into account the multiple classes, therefore the alignment v is now transformed into a vector $\mathbf{v}$. Intuitively, $v_\ell = 1$ means that $\varphi_{j,q}(\mathbf{x})$ agrees with (or votes *for*) the ℓth class, and $v_\ell = -1$ means that it disagrees with (or votes *against*) it. The decision stumps function (2.22) that minimizes the weighted error is now

$$\left(j^{(t)}, q^{(t)}, \mathbf{v}^{(t)}\right) = \arg\min_{j,q,\mathbf{v}} \epsilon\left(\varphi_{j,q}(\cdot)\mathbf{v}\right),$$

and the optimal weak classifier, which became a vector $\mathbf{h}^{(t)} : \mathbb{R}^d \to \{-1, 1\}^k$, as

$$\mathbf{h}^{(t)}(\cdot) = \varphi_{j^{(t)},q^{(t)}}(\cdot)\mathbf{v}^{(t)}. \tag{2.26}$$

The weighted misclassification error of (2.1) is now computed over all the classes

$$\epsilon^{(t)} = \epsilon\left(\mathbf{h}^{(t)}(\cdot)\right) = \sum_{i=1}^{n}\sum_{\ell=1}^{k} w_{i,\ell}^{(t)} \mathbb{I}\left\{h_\ell^{(t)}(\mathbf{x}_i) \neq y_{i,\ell}\right\}. \tag{2.27}$$

We call this strategy *single-threshold* because the class-wise binary decisions $h_1^{(t)}, \ldots, h_k^{(t)}$ share the decision-stump threshold $q^{(t)}$.

We consider also a second, *multi-threshold* strategy, where the class-wise binary decisions still share the decision-stump attribute j as for the single-threshold, but for each class ℓ, we select a different threshold $q_\ell^{(t)}$. Formally, we search over the features $j = 1, \ldots, d$, threshold vectors $\mathbf{q}$, and alignment vectors $\mathbf{v}$ to obtain

$$\left(j^{(t)}, \mathbf{q}^{(t)}, \mathbf{v}^{(t)}\right) = \arg\min_{j,\mathbf{q},\mathbf{v}} \sum_{i=1}^{n}\sum_{\ell=1}^{k} w_{i,\ell}^{(t)} \mathbb{I}\left\{\varphi_{j,q_\ell}(\mathbf{x}_i)v_\ell \neq y_{i,\ell}\right\}.$$

In the rest of the work, we will refer to the single and multi-threshold version as AB.SINGLE and AB.MULTI. Figure 2.4 shows a simple example of the differences among the two approaches. The multi-class version of the tree algorithm described in Section 2.3.2 can use either the single or the multi-threshold approach, as each node of the tree behave like a simple decision stumps. In this work we will refer to AB.TREE as the single-threshold version.

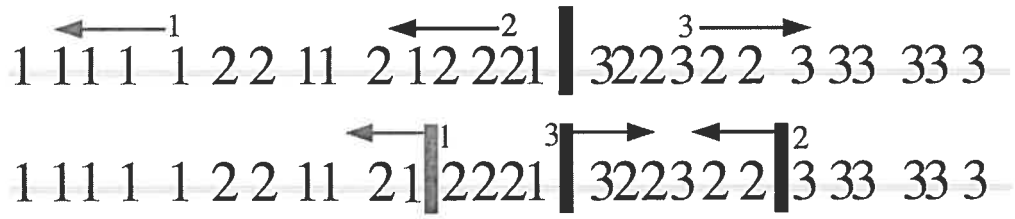


Figure 2.4: The two different ways of treating the multiclass problem with stumps. In the first one on the top (AB.SINGLE) there is just a single threshold, which is selected by minimizing the overall weighted error. In the bottom one (AB.MULTI), each class finds its own threshold, computed by comparing the class against all of them. The arrows represent the elements of the $\mathbf{v}$ vector.

The alignment vector $\mathbf{v}$ can be optimized element-wise in the following way. For a given binary classifier[4] φ, and for all classes $\ell = 1, \ldots, K$, we define

$$\begin{aligned} \mu_{\ell-} &= \sum_{i=1}^{n} w_{i,\ell}^{(t)} \mathbb{I}\left\{\varphi(\mathbf{x}_i) y_{i,\ell} = -1\right\}, \\ \mu_{\ell+} &= \sum_{i=1}^{n} w_{i,\ell}^{(t)} \mathbb{I}\left\{\varphi(\mathbf{x}_i) y_{i,\ell} = +1\right\}, \\ \mu_{\ell 0} &= \mu_{\ell-} + \mu_{\ell+}. \end{aligned} \tag{2.28}$$

[4]For simplicity we omit $_{j^{(t)},q^{(t)}}$ and $^{(t)}$.

With this notation, for a given vote vector $\mathbf{v} = (v_1, \ldots, v_K)$, it is clear that

$$\begin{aligned}
\epsilon_- &= \sum_{\ell=1}^{K} \Big(\mu_{\ell-} \mathbb{I}\{v_\ell = +1\} + \mu_{\ell+} \mathbb{I}\{v_\ell = -1\} \Big), \\
\epsilon_+ &= \sum_{\ell=1}^{K} \Big(\mu_{\ell+} \mathbb{I}\{v_\ell = +1\} + \mu_{\ell-} \mathbb{I}\{v_\ell = -1\} \Big), \\
\epsilon_0 &= \sum_{\ell=1}^{K} \mu_{\ell 0} \mathbb{I}\{v_\ell = 0\}.
\end{aligned}$$

If abstention is not used, the values of the alignment vector are simply

$$v_\ell = \begin{cases} 1 & \text{if } \mu_{\ell+} > \mu_{\ell-} \\ -1 & \text{otherwise.} \end{cases} \tag{2.29}$$

Both strategies AB.SINGLE and AB.MULTI share the same complexity, which is $O(kdn)$. This assuming that the feature vector projections $(x_1^{(j)}, \ldots, x_n^{(j)})$ are ordered beforehand. If this is not true, the sorting time brings the complexity to $O(kdn \log n)$. The cost of the whole algorithm is respectively $O\big(nd(kT + \log n)\big)$ and $O\big(n \log ndkT\big)$ times.

Once the weak classifier is selected, the coefficient α is computed in the same way as (2.2). In the final step of each iteration, we re-weight the data points using the formula

$$w_{i,\ell}^{(t+1)} = \frac{w_{i,\ell}^{(t)} \exp\Big(-\alpha^{(t)} y_{i,\ell} h_\ell^{(t)}(\mathbf{x}_i)\Big)}{Z^{(t)}},$$

where $Z^{(t)}$ is the usual normalization factor such that $\sum_i^n \sum_\ell^k w_{i,\ell}^{(t+1)} = 1$.

After T iterations, the algorithm outputs a vector-valued discriminant function

$$\mathbf{g}(\mathbf{x}) = \sum_{t=1}^{T} \alpha^{(t)} \mathbf{h}^{(t)}(\mathbf{x}).$$

To obtain a single label, we simply take the class that receives the "most vote",

that is,

$$f(\mathbf{x}) = \arg\max_{\ell} g_\ell(\mathbf{x}).$$

Algorithm 3 AdaBoost.MH

Input: D_n, $\textsc{Weak}(D_n, \mathbf{w}^{(1)})$, T

for $i \leftarrow 1$ to n **do**
  for $\ell \leftarrow 1$ to k **do**
    $w_{i,\ell}^{(1)} \leftarrow \begin{cases} \frac{1}{2n} & \text{if } y_{i,\ell} = 1, \\ \frac{1}{2n(k-1)} & \text{otherwise.} \end{cases}$ {Initialize weights}
  end for
end for
for $t \leftarrow 1$ to T **do**
  $\mathbf{h}^{(t)} \leftarrow \textsc{Weak}(D_n, \mathbf{w}^{(t)})$ {Find optimal weak classifier}
  $\epsilon^{(t)} \leftarrow \sum_{h_\ell^{(t)}(\mathbf{x}_i) \neq y_{i,\ell}} w_{i,\ell}^{(t)}$ {Compute the weighted error}
  $\alpha^{(t)} \leftarrow \frac{1}{2} \ln\left(\frac{1-\epsilon^{(t)}}{\epsilon^{(t)}}\right)$ {Compute the confidence}
  for $i \leftarrow 1$ to n **do**
    for $\ell \leftarrow 1$ to k **do**
      $w_{i,\ell}^{(t+1)} \leftarrow \frac{w_{i,\ell}^{(t)} \exp\left(-\alpha^{(t)} y_{i,\ell}\, h_\ell^{(t)}(\mathbf{x}_i)\right)}{Z^{(t)}}$ {Re-weight the points}
    end for
  end for
end for
return $f_T(\cdot) = \arg\max_{\ell} \sum_{t=1}^{T} \alpha^{(t)} h_\ell^{(t)}(\cdot)$ {Combined classifier}

By sorting $f(\mathbf{x})$ we can obtain a simple rank of the classes, which can be useful in the situations where an example has not well defined class (borderline) or it belongs to many classes, as it is often the case with music. The described method is summarized in Algorithm 3.

The extension of AdaBoost.MH to regularization does not require any particular modification of the algorithm as only $\alpha^{(t)}$ and the early stopping criterion have to be changed.

In this work we considered also abstention. We do not leave the decision to the weak learner as usual, but we add 0 to the decisions of the vote vector $\mathbf{v}$. This is done by optimizing (2.26) with a greedy algorithm (that, for the time being, is

not proved to be optimal in terms of (2.27)). We first set $\mathbf{v}$ as in (2.29). Then, in an iteration over the labels, we select the "best" element of $\mathbf{v}$ to set to 0, that is, the one that decreases (2.27) the most. The algorithm stops when (2.27) does not decrease anymore.

2.5.3 AdaBoost.MO

In the same article in which they suggested the previous algorithm, Schapire and Singer proposed also another variation called **AdaBoost.MO**, in which the multiclass problem is, again, partitioned into a set of binary problems. This approach was first proposed by Dietterich and Bakiri in 1995 [DB95] in a somewhat different setting. They suggest using error correcting output codes (ECOC), which they consider to be designed to separate classes far from each other in terms of their symmetric difference, to map the label on binary sets.

Intuitively we can say that if we have a set where $\mathcal{Y} = \{$*apple, orange, cherry, carrot, potato*$\}$ we can separate the points by mapping them into the binary representation $\mathcal{Y}' = \{$*fruit, vegetable*$\}$. The more the sets are separated from each other (apples are *closer* to oranges than potatoes), the better the classification is likely to be.

More formally let $H : \mathcal{X} \to \mathcal{Y}$ and $\mathcal{Y} = \{y_1, ..., y_k\}$ be our k-polychotomy. The decomposition of the k-polichotomy generates a set of L mapping functions $\lambda_1, ..., \lambda_L$, where each subdivide the input pattern in two complementary sets (or superclasses) $\mathcal{Y}_j^+$ and $\mathcal{Y}_j^-$. These sets contain each one or more classes of the original k-polychotomy, as showed in Figure 2.5, thus the mapping[5] is defined as

$$\lambda_j(y_k) = \begin{cases} +1 & \text{if } y_k \subseteq \mathcal{Y}_j^+, \\ -1 & \text{if } y_k \subseteq \mathcal{Y}_j^-. \end{cases}$$

Note that λ maps to subsets of an unspecified label set $\mathcal{Y}'$ which does not need to

[5]We could also use ternary codes, i.e. $\{-1, 0, +1\}$, allowing abstention.

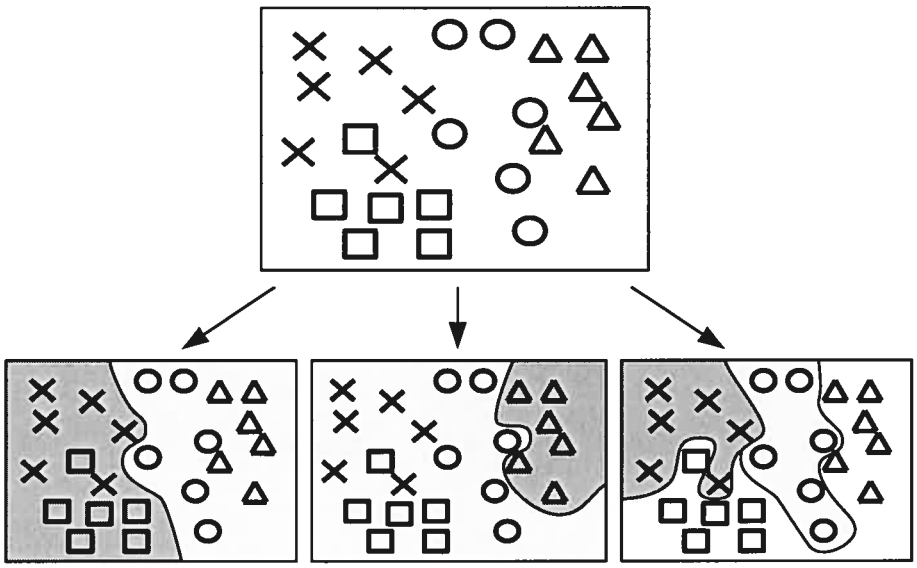


Figure 2.5: A four classes problem mapped using three different λ functions. In the first mapping the classes are balanced on the two superclasses. The second is a one-vs-all mapping, where the third one, assuming 0 values in ternary codes, is similar to a one-vs-one approach.

be the same as $\mathcal{Y}$.

Consider, for instance, a decomposition for a 4 class classification problem into seven possible mappings (defined by Dietterich and Bakiri as a *7-bit class coding*), as showed in Table 2.1. The task of the second mapping, namely λ_2, consists in separating the patterns belonging to class y_1 and y_4 from the patterns of class y_2 and y_3.

The columns of the table represent a *codeword*. For instance, associated to class y_3 is the codeword $[-1, -1, +1, +1, -1, -1, +1]$. ECOC decomposition tries to maximize error recovering capabilities through the maximization of the minimum distance between each couple of codewords. For further information on ECOC see also [MV00].

Schapire and Singer, in their work, suggest that when then number of classes mapped into a set is large enough (that is $k' = |\mathcal{Y}'|$) is not too small), we can expect

Table 2.1: An example of ECOC decomposition matrix with a 7-bit class coding, on a 4 classes problem.

Mapping function	Classes (y_ℓ)			
	y_1	y_2	y_3	y_4
λ_1	+1	-1	-1	-1
λ_2	+1	-1	-1	+1
λ_3	+1	-1	+1	-1
λ_4	+1	-1	+1	+1
λ_5	+1	+1	-1	-1
λ_6	+1	+1	-1	+1
λ_7	+1	+1	+1	-1

to get a similar effect by choosing a λ_j entirely at random, so that for every set $\mathcal{Y}'$ we include a class ℓ with equal probability. However Allwein et al. [ASS00] showed that the choice of the mapping function λ might be crucial. They demonstrated that after many experiments no output code is clearly better, but poorly chosen code can lead to unnatural binary problems that are hard to learn. They concluded that the choice of the best code depends on the domain.

Once the mapping function λ has been found we can apply AdaBoost.MH directly on each binary function $h : \mathcal{X} \rightarrow \{-1, +1\}$ with the transformed training data $(\mathbf{x}_i, \lambda(y_i))$. The classification of the new instance x can be done in two different ways. The first one, namely the one proposed by Dietterich et al., is to evaluate H on $\mathbf{x}$ to obtain a set $H(\mathbf{x}) \subseteq \mathcal{Y}'$. Then we choose the label $y \in \mathcal{Y}$ for which the mapped output code $\lambda(y)$ has the shortest Hamming distance to $H(\mathbf{x})$.

The Hamming distance is simply the number of positions in two strings of equal length for which the corresponding elements (in our case -1 an 1) are different.

Formally, we choose:

$$\arg\min_{y \in \mathcal{Y}} |\lambda(y) \Delta H(\mathbf{x})|,$$

where Δ is the Hamming-distance operator.

A problem of this approach is that it ignores the confidence with which each label was, or was not, included in the final hypothesis $H(\mathbf{x})$. Schapire and Singer,

indicate an alternative approach in which the goal is to predict the label y which, if it had been paired with $\mathbf{x}$ in the training set, would have caused the pair $(\mathbf{x}, y)$ to be given the smallest weight, thus the better confidence, under the final distribution. Formally, the chosen class is

$$\arg\min_{y \in \mathcal{Y}} \sum_{y' \in \mathcal{Y}'} \exp\left(-\lambda_{y'}(y) f(\mathbf{x}, y')\right)$$

where $f(\mathbf{x}, y') = \sum_t \alpha^{(t)} h^{(t)}(\mathbf{x}, y')$, that is the linear sum of weak learners before the remapping.

Algorithm 4 shows the pseudocode of AdaBoost.MO. Note that the function λ returns different mapping on different labels y, and that $\lambda(y)$ is a vector.

Algorithm 4 AdaBoost.MO

Input: D_n, $\lambda : \mathcal{Y} \to 2^{\mathcal{Y}'}$, AdaBoost.MH

1: Run AdaBoost.MH on relabeled data: $(\mathbf{x}_1, \lambda(y_1)), ..., (\mathbf{x}_n, \lambda(y_n))$
2: Get back final hypothesis H in the form $H(\mathbf{x}, y') = \text{sign}(f(\mathbf{x}, y'))$
   where $f(\mathbf{x}, y') = \sum_t \alpha^{(t)} h^{(t)}(\mathbf{x}, y')$
3: **return** (variant 1) $H_1(\mathbf{x}) = \arg\min_{y \in \mathcal{Y}} |\lambda(y) \Delta H(\mathbf{x})|$
4: **return** (variant 2) $H_2(\mathbf{x}) = \arg\min_{y \in \mathcal{Y}} \sum_{y' \in \mathcal{Y}'} \exp\left(-\lambda_{y'}(y) f(\mathbf{x}, y')\right)$

2.6 Other Classifiers

2.6.1 Support Vector Machines

Support Vector Machines (SVMs) were first described by Vapnik [Vap99]. The principle is to use a *kernel-function* that computes the similarity between pairs of training examples, and then to choose some training examples (the support vectors) as a basis for a binary classification. [XCST03] used a kernel based on a wavelet decompositions of the two input songs.

Mandel and Ellis [ME05b] used a kernel based on the KL-divergence between mixtures of Gaussians in the feature space. In these works, multiclass classification is performed using multiple *one-versus-all* SVMs.

2.6.2 Artificial Neural Networks (ANNs)

Artificial neural networks (ANNs) use gradient descent to minimize a cost function by adjusting weighted connections between hidden units. When the hidden units are nonlinear and bounded (usually sigmoidal) an ANN is a very powerful function approximator. ANNs have been used extensively for decades in machine learning tasks like classification and regression [Bis95]. For classification tasks, it is customary to train the network to maximize the probability of the training data under the distribution output by a softmax function. The regularization method that we adopted in Section 4.3 was a small random-weight initialization, followed by early stopping. For our experiments we stopped gradient descent heuristically, when 50 iterations of minimization failed to yield an improved model according to a held-out validation set.

For more on SVMs and ANN, see [DHS01].

2.7 Conclusion

In this chapter we have mainly focused our attention on the algorithm used for our learning tasks, that is AdaBoost. In Section 2.1 we briefly discussed the history of this algorithm, and we have showed the details of its implementation in the following section (2.2). We discussed two extensions of the algorithm, abstention and regularization, in Section 2.4, and two approaches to modify AdaBoost in order to solve multi-class problems in Section 2.5. Finally, in Section 2.6 we briefly discussed two other learning algorithms, SVM and ANN, which we are using for comparison in Section 4.3.

CHAPTER 3

AUDIO FEATURE EXTRACTION

To learn from examples, an algorithm must be fed data. Digitalization is a common way of transforming the alternation in pressure propagated in the air that we call "sound", into a sequence of discrete values (samples) interpretable by a computer. These samples are then quantized and encoded with various levels of sophistication into file formats such as the common `wav` (`WAVE`) by Microsoft or `au` by Sun.

However, in music information retrieval, there is a general consensus among the published methods that using raw signal (the simplest level of encoding) is not a viable approach, and this is for several reasons:

- Raw signals are very large vectors. The size of these vectors does not only depend on the length of the song, but also on the frequency of sampling rate. To capture rhythmical data, or even just to recognize an instrument, hundreds thousands of samples needs to be evaluated.

- Classification should be completely invariant to global shifting of the signal (e.g. a delay by one sample).

- There are many perceptually significant events that are hidden in the raw waveform, such as small differences in small magnitudes of spectral components (for instance the slight change in one of the harmonics in frequency space) or the relative changes in pitch between two instruments, that are very difficult to detect.

- On the other hand, some events might look significant in the raw signal, but are not important perceptually, like global changes in pitch.

Instead of taking the raw signal, a successful algorithm must extract a number of acoustic features which capture various salient aspects of music. For instance

one of the most used techniques is a time-space transform via the Discrete Fourier Transformation (DFT). Another common strategy is to simulate the human audition system in order to better match the energy patterns of the sound to what we really "hear". Techniques that do this include Linear Predictive Coefficients (LPC), Mel-frequency cepstral coefficients (MFCC) and other algorithms that we will discuss in this chapter. In general, music information retrieval techniques use knowledge from signal processing theory, physics of sound, psychoacoustics, human auditory perception and music theory.

A recent survey by Aucouturier and Pachet [AP03] gives a comprehensive list of features used for music information extraction, although research in feature-extraction continues.

3.1 Frame Level Timbral Features

Li and Tzanetakis [LT03b] defines three types of auditory features: the timbral texture, the rhythm, and the pitch. Timbre is the sound-quality of near-instants, or simply the quality of a musical sound, which distinguishes different types of sound production or instruments. Rhythm is the variation and duration of patterns of sounds in time. Finally pitch is the psychoacoustic perception of the frequency of a sound (i.e. a note). In this section we describe features that aim to capture the first type, while in Section 3.3 we briefly review two methods to extract rhythm. We do not discuss pitch as it is extremely hard to capture in polyphonic sounds, and current methods are fairly unreliable.

In the domain of timbre-extraction we define a *frame* as a real-valued vector $\mathbf{u}$ of length τ, that corresponds to around a twentieth of a second. At that scale, acoustic signals are effectively periodic. All the features below, with the exception of the zero-crossing rate and linear predictive coefficients, are defined in terms of the absolute value of the discrete Fourier transform, $\mathbf{F}$ of $\mathbf{u}$, which is derived from the Fourier transform.

3.1.1 FFTC

This feature is simply the magnitude of the coefficients of the Fourier transform. In theory these coefficients indicate energy present in each frequency whose wavelength fits an integer number of times into the sample. In practice these coefficients are much more noisy due to aliasing artifacts inherent in looking at such a small sample.

3.1.2 RCEPS

Real cepstral coefficients are defined as:

$$\begin{aligned} L^{(i)}_{rceps} &= \log(F^{(i)} + \gamma), \\ \mathbf{r}_{rceps} &= \mathrm{dct}(\mathbf{L}_{rceps}). \end{aligned}$$

Within, γ is a measure of uncertainty or noise in the components of $\mathbf{F}$, log maps narrow-band power in the signal to narrow-band loudness, and dct is the discrete cosine transform which is equivalent to a discrete Fourier transform followed by a projection that discards all imaginary components. The variable i indexes frequencies. These coefficients estimate the strength of different harmonic series in the signal, but they are quite sensitive to noise. For classifiers that learn general linear transforms of the input, the discrete cosine transform is unnecessary and potentially harmful.

3.1.3 MFCC

Mel-frequency cepstral coefficients are computed almost the same way as RCEPS, except that the input $\mathbf{F}$ is first projected according to the Mel-scale. The Mel-scale is a psycho-acoustic frequency scale on which a change of 1 unit carries the same perceptual significance, regardless of the position on the scale. This transform from a linear (Hertz) scale is motivated by human auditory perception: while humans are very capable of differentiating 440Hz from 441Hz, they are not capable of dif-

ferentiating 4400Hz from 4401Hz. The Mel-scale increases identically with Hertz from 0 to 1000Hz, at which point it continues to rise logarithmically. We denote the Mel-scale projection of $\mathbf{F}$, $\mathbf{F}_{Mel}$; while $\mathbf{F}$ is often quite sparse in the high-frequency range, energy is distributed more uniformly over $\mathbf{F}_{Mel}$.

Mel-frequency cepstral coefficients are defined as:

$$\begin{aligned} L_{mfcc}^{(i)} &= \log(F_{Mel}^{(i)} + \gamma), \\ \mathbf{r}_{mfcc} &= \mathrm{dct}(\mathbf{L}_{mfcc}). \end{aligned}$$

Each coefficient is best conceived as an abstract measure of the shape of L, and as with RCEPS, for some classifiers, the dct is unnecessary.

3.1.4 ZCR

The zero-crossing rate is

$$r_{zcr} = \frac{1}{\tau} \sum_{i=1}^{\tau} \mathbb{I}\left\{u^{(i)} u^{(i-1)} < 0\right\}.$$

This function is simply the number of times the signal changes sign. It gained popularity as a computationally cheap way to distinguish sibilants and other consonants from vowels in order to quickly segment speech signals. Other authors have described it as a rough estimate of the noisiness of the signal.

3.1.5 Rolloff

The rolloff feature is the spectral boundary of the lower b fraction of the total energy in $\mathbf{F}$, measured from 0Hz upward.

$$r_{ro,b} = \max\{y | b > \sum_{i=1}^{y} F^{(i)}\}.$$

We computed the rolloff at 16 equally spaced thresholds between 0 and 1 (exclusive). Due to the sparseness of energy in F, this feature might be more reasonably

applied to F_{Mel}, but we did not try this.

3.1.6 Spectral Centroid

The spectral centroid is the weighted-mean spectral component.

$$r_{cent} = \frac{\sum_i F^{(i)} i}{\sum_i F^{(i)}}.$$

Here $F^{(i)}$ denotes the i'th component of $\mathbf{F}$. The spectral centroid has been described as a measure of the brightness of the tone.

3.1.7 Spectral Spread

While the spectral centroid is like the mean frequency, the spectral spread is the variance.

$$r_{spread} = \frac{\sum_i F^{(i)} (i - r_{cent})^2}{\sum_i F^{(i)}}.$$

where, again, $F^{(i)}$ denotes the i'th component of $\mathbf{F}$.

3.1.8 Autoregression (LPC)

The w linear predictive coefficients of $\mathbf{u}$ are:

$$\begin{aligned} \mathbf{r}_{lpc} &= \arg\min_{\mathbf{b}} \sum_{i=1}^{\tau} (u^{(i)} - \sum_{j=1}^{w} b^{(j)} u^{(i-j)})^2, \\ r_{lpcerr} &= \min_{\mathbf{b}} \sum_{i=1}^{\tau} (u^{(i)} - \sum_{j=1}^{w} b^{(j)} u^{(i-j)})^2. \end{aligned}$$

These can be computed efficiently from the signal's autocorrelation (see also Section 3.3.1) by the Levinson-Durbin recursion [Mak75]. The linear predictive coefficients are another phase-independent measure of the harmonic content of a signal.

In general, a feature extractor is a vector-valued function $r(\mathbf{u})$ from $\mathbb{R}^m$ to $\mathbb{R}^n$, in which $\mathbf{u}$ is a signal of length m, and n may be the sum of some number of

MFCC and RCEPS coefficients, for instance. It is conceivable that some features only make sense in certain contexts, such as note onsets or instrumental solos, but we will consider that r can be reasonably applied to any block of m contiguous samples from the signal.

3.2 Blending frame-level audio features

Even the features described in the previous sections cannot be used directly as input to the learning algorithm, as their dimensionality (the number of components of the feature) is too large even on short recordings. A standard four minute song (240 seconds) at 44.1kHz, is a vector $\mathbf{u} \in \mathbb{R}^p$ where $p = 240 \times 44,100 = 10,584,000$ samples. The features are calculated over short frames of music, generally between 20ms and 100ms, which means respectively 882 to 4410 samples per frame. At that rate we would need 2400 frames for the 100ms features in order to evaluate the whole song, and this with just a single feature type!

It is of course possible to downsample the signal, but there is a practical limit to downsampling for timbre features given that much of the information for timbre is found at high frequencies. Using large windows is not practical as some features like the MFCC are most effective when applied to spectrally-stable near-instants of sound. In fact, if the windows are too large, the probability of blending two or more timbres into the same feature becomes very high.

Standard dimensionality reduction algorithms, such as PCA (see [DHS01] page 568), Isomap [TdSL00] or kernel based PCAs, are often used, especially if little is known about the data. In theory, these methods should make it possible to capture the maximum amount of information, but in practice the resulting feature space is not easily interpretable, and if important distinctions are in dimensions with low variance they risk being lost, as these methods are unsupervised.

Recently, West and Cox [WC05] used a segmentation strategy to address this issue. They used an onset detection algorithm to partition the song into a set of segments that correspond to instruments playing notes. Our approach is similar

but simpler. We also partition the song into pieces, but the size of the segments is constant. Each segment is classified by AdaBoost.MH which returns a real number for each class representing a vote. We then sum the vector $\mathbf{h} \in \mathbb{R}^k$ obtained for each segment and take a majority vote. This has the advantage of both reducing the size of the features vector, and concentrating more on local aspects of the song. We explain this approach more in details in Section 4.3.

3.3 Auditory Temporal Features

The features discussed so far are common strategies to extract timbral information from the soundfile. They are very useful, for example, to distinguish the high vibrato of a trumpet from the low-pitched sound of a bass, or to detect the roughness of a given sound. However, at a higher level, temporal structures are clearly defining the real essence of music; ignoring these structures could be a mistake. In fact, music seems to involve a subtle balancing game between expectation and surprise. The first represents the cyclic aspect of sound and can be hidden, to delight our playful mind, inside ritornellos, hierarchical structures, inverse phrases, and many others schemas. The role of expectation in music seems unique among the arts. As singer-songwriter Joni Mitchell quipped:

> *No one nobody ever said to Van Gogh, "Paint us Starry Night again, man!"*

On the other hand, the surprise is what breaks the cycle, perhaps making music more enjoyable.

In this section we discuss two similar approaches which try to capture one basic component of expectation: musical tempo.

3.3.1 Autocorrelation

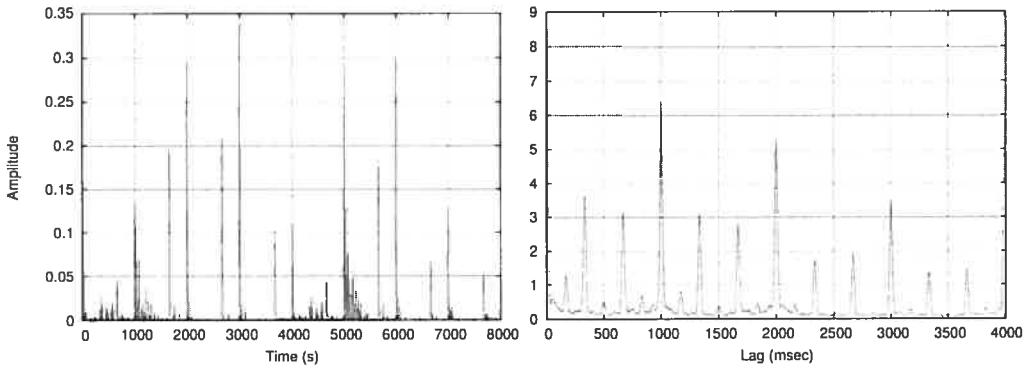
Autocorrelation is a powerful tool to find simple time-repeating patterns inside a signal. It works by transforming the input from its time domain representation

into the frequency domain and back again, discarding phase. Formally, it is the cross-correlation of a signal $\mathbf{u}$ with itself, and it is defined as

$$a^{(l)}(\mathbf{u}) = \sum_{i=0}^{\tau-l} u^{(i)} u^{(i+1)} \tag{3.1}$$

where l is the lag.

The autocorrelation is generally computed for a range $\{l_1, \ldots, l_n\}$ of lags, in which the peaks represent recurring events in the signal and where high correlation exists, as showed in Figure 3.1. A random signal, on the other hand, should be roughly equal across all the lags.



(a) Original signal (b) Four seconds autocorrelation

Figure 3.1: The signal (a) represent the first 8 second of the *Blue Danube* waltz, rendered from a MIDI source into a WAV file. The autocorrelation (b) clearly shows a strong spike at one second lag, and two smaller every third and two thirds of a second, as expected with the 3/4 metric of the waltz.

There is also an important relationship between the autocorrelation and the Fourier Transform known as the Wiener-Khinchin theorem, which allows us to compute the autocorrelation in time $O(n \log n)$ using a DFT instead of $O(n^2)$. The formula (3.1) can therefore be changed into

$$\mathbf{a}(\mathbf{u}) = \text{IDFT}(|\mathbf{F}|)$$

where IDFT is the *Inverse* Discrete Fourier Transformation, and || is the complex modulus.

There are several simple methods to find the tempo among the spikes of the autocorrelation. One option is to pick the lag closest to a comfortable tapping rate, say 600 ms. A second better option is to multiply the autocorrelation lags by a window such that more accent is placed on lags near a preferred tapping rate. A function with such properties is the Gaussian window centered at 600ms and symmetrical in log-scale frequency, proposed by Parncutt [Par94] by his studies in human perception. Even better results are had if lags having low-order integer ratios relating to meter structure are used [EC05].

3.3.2 Phase Preserving Autocorrelation

While the autocorrelation is widely used as feature to discover rhythmical structures, it has many drawbacks. Autocorrelation can give us useful information about something repetitive happening every 2 seconds, but cannot tell *when* this is happening because the phase information is lost in the computation. A possible solution is to keep the phase information stored elsewhere and use it to track the chosen lag in the original signal.

A bigger problem with autocorrelation arises when analyzing signal that lacks of strong onset energy, which is common with voice or in some kinds of music such as classical where we have smoothly changing instruments like strings. In this cases autocorrelation tends to be flat, and it is very difficult to distinguish meaningful peaks. See for example a song from Manos Xatzidakis from the ISMIR 2004 Tempo Induction in Figure 3.2. Not only are the peaks less sharp but they are poorly aligned with the target tempo.

A common solution is to apply the autocorrelation to a number of band-pass filtered versions of the signal [Sch98]. This helps generally on noisy polyphonic input, where the hope is to capture "beating" instruments hidden in particular bands of frequencies.

An approach that has been recently proposed by Eck in collaboration with the

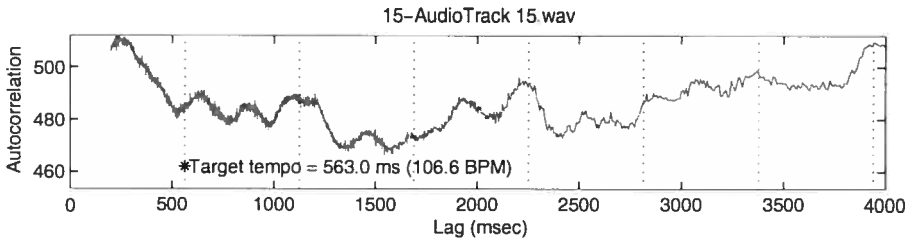


Figure 3.2: Autocorrelation of a song by Manos Xatzidakis from the ISMIR 2004 Tempo Induction contest. The dotted vertical lines mark the actual tempo of the song and harmonics of the tempo.

author [EC05], solves both problems, the flatness and the misalignment, at the same time. Instead of computing a simple autocorrelation for a fixed phase, we create a matrix in which the signal is "wrapped", via modulo arithmetic to store intermediate results, for every lag and phase ϕ

$$A^{(l)}(\mathbf{u}) = \left(\sum_{i=0}^{\lfloor \frac{\tau - l}{l} \rfloor} u^{(li+\phi)} u^{(l(i+1)+\phi)} \right)_{\phi=0}^{l-1}$$

The matrix can be easily updated in real-time, by multiplying the sample input with the modulo of each lag. In Figure 3.3 we can see this matrix of the first pattern from the set found in Povel and Essens [PE85]. The pattern itself is visible at the top of the matrix, at the lag corresponding to its period. At lower lag we observe structures that indicate the hierarchy of the sound, where we can also find the peaks over the true tempo of the song.

It is interesting to see that when we sum each lag of the matrix, we still obtain the autocorrelation. By doing so, all the structural information that it is visible in our matrix is lost in the process, and this is probably the main reason why autocorrelation fails to capture the true beats, even if they are perceptually present.

However, there is a simple way to exploit the autocorrelation phase matrix to improve significantly the performance of autocorrelation. The idea is that

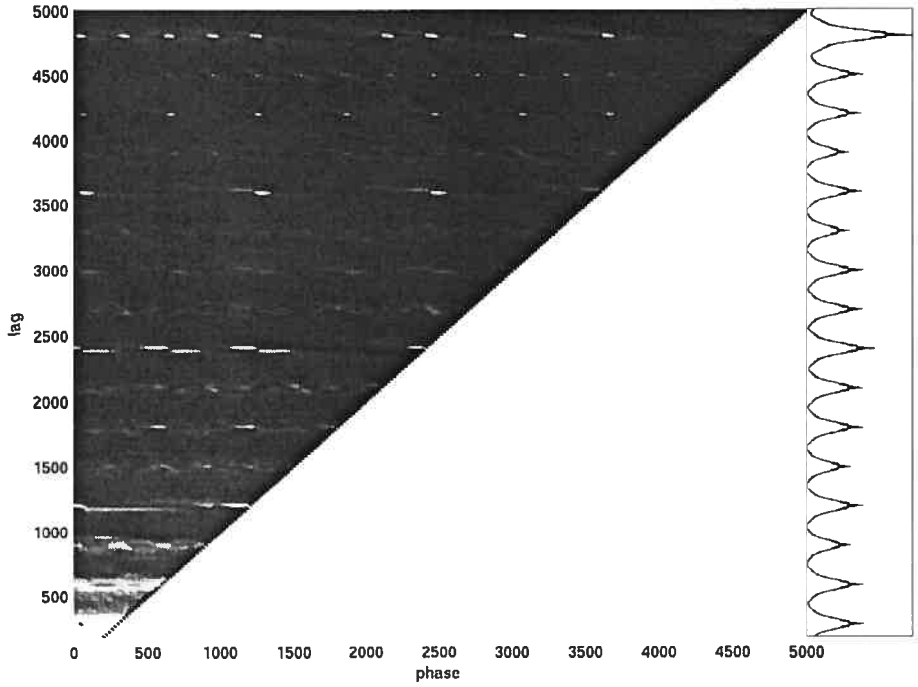


Figure 3.3: The autocorrelation phase matrix for the first pattern of Povel & Essens, computed for lags 250ms through 500ms. If we sum each lag we obtain the current autocorrelation of the input signal (seen at right).

structurally-salient lags will tend to have more "spike-like" distributions than non-structural lags. Thus even if the *autocorrelation* is evenly distributed by lag, the distribution of *autocorrelation energy in phase space* should not be so evenly distributed.

To measure the "structureness" of our signal, we use the Shannon entropy formula. This formula gives an evaluation of the amount of "disorder" in a system, and it is defined as

$$S(\mathbf{x}) = -\sum_{i=1}^{\tau} x^{(i)} \log_2(x^{(i)})$$

where $\mathbf{x}$ is a probability function, where $x^{(i)} \geq 0$ and $\sum x^{(i)} = 1$.

We compute the entropy for lag l in the autocorrelation phase matrix as follows:

$$A_{sum}^{(l)} = \sum_{i=1}^{\tau} A^{(l,i)}$$

$$S^{(l)} = -\sum_{i=1}^{\tau} \frac{A^{(l,i)}}{A_{sum}^{(l)}} \log_2 \left(\frac{A^{(l,i)}}{A_{sum}^{(l)}} \right)$$

We then multiply this entropy value into the autocorrelation, after having scaled both to a range between 0 and 1. This simple solution significantly improves tempo induction. For instance, in Figure 3.4 we can see the autocorrelation showed in Figure 3.2 multiplied with the computed entropy. The spikes align perfectly with the target lag and its multiples, where previously such information was missing.

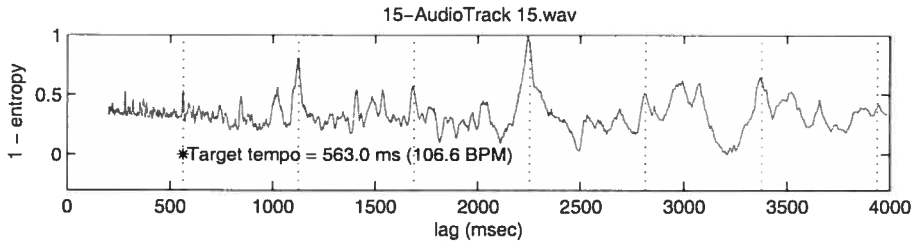


Figure 3.4: The entropy calculation of the same Manos Xatzidakis song shown in Figure 3.2. The plot displays the normalized opposite of entropy for clarity. Observe how the entropy spikes align well with the correct tempo lags and with its harmonics.

Although it is not the main focus of this work, as we will not use this feature for our classification tasks, this approach can be used for effective meter discovery. By simply summing hierarchical combinations of autocorrelation lags, we can get a rough estimate of the metric of the signal. The likelihood of a duple meter M_{duple} existing at lag l can be estimated using the following sum:

$$M_{duple}^{(l)} = AE^{(l)} + AE^{(2l)} + AE^{(4l)} + AE^{(8l)}$$

and the likelihood of a triple meter is estimated using the following sum:

$$M^{(l)}_{triple} = AE^{(l)} + AE^{(3l)} + AE^{(6l)} + AE^{(12l)}$$

Other candidate meters can be constructed using similar combinations of lags. The winning meter will be chosen by sampling all reasonable lags (for instance 200ms $\leq l \leq$ 2000ms) and then comparing the resulting $M^{(l)}_{*}$ values. Provided that the same number of points are used for all candidate meters, the comparison can be done directly. This search can be done in a efficient way, as each lag/candidate meter requires only a few additions.

The phase preserving autocorrelation is an effective and robust method that does not involve training to find tempo and the metrical structure of a signal. In [EC05] we applied it on two dataset from the ISMIR 2004 Tempo Induction contest, and the performance was comparable to the contest winner.

3.4 Haar-like Features

In this section we discuss a type of procedural[1] feature, that has been successfully used in computer vision to address high dimensional data in a robust and fast fashion. Although we are working with sounds, it is possible to render them as 2d images (i.e. with a sequence of short timescale Fourier transformations (STFT), known as spectrogram), and exploit this representation to find patterns which can be used for to classify the signal [CEK05b]. In Section 4.2 we discuss how to use these features to build a robust frame-level music/speech discriminator, while here we address the basic theory of the Haar-like features.

In computer vision it is a common problem that the input data has a large number of dimensions. A simple VGA image of size 640 × 480 has a total of 307,200 pixels, each of which is a dimension. Clearly, the number of observations

[1] With *procedural* we refer to a certain type of feature that for some reason (technical, computational or algorithmic) cannot be pre-computed and is instead obtained as the algorithm needs it.

that one would need to learn in such large space is too large for any real-world application. It seems foolish, however, to evaluate each single dimension of the image when we know that the object is characterized by few particular attributes within groups of pixels. In fact, it is an error to consider a pixel as a totally independent entity, as they are very likely correlated with the neighbors in the two dimensional space in which they lie. It makes sense, with this premise, to work with regions of pixels instead of single elements, because of the higher probability of capturing the meaningful information we need for classification.

The *Haar wavelets*, first proposed by Papageorigiu et al. [POP98], are a natural set of basis functions which encode differences in average intensities between different regions within an image. Figure 3.5 shows four possible configurations of these wavelets.

Figure 3.5: Four possible configurations of the Haar wavelets. The first two are aimed to capture vertical and horizontal patterns, the third is looking for "holes", and the fourth should get diagonal patterns.

The principle is simple: each feature is composed of group of rectangular black or white blocks, with different sizes and location within a defined area. Formally, for a black block with its upper left corner placed at (x, y), and with size $w_x \times w_y$, of image I we compute the function

$$B^{(x,y,w_x,w_y)}(I) = \sum_{i=x}^{x+w_x} \sum_{j=y}^{y+w_y} I^{(i,j)}, \tag{3.2}$$

which is simply the sum of the intensity of the image in the black area. For a white

block, we just compute the negative function

$$W^{(x,y,w_x,w_y)}(I) = -B^{(x,y,w_x,w_y)}(I) = -\sum_{i=x}^{x+w_y}\sum_{j=y}^{y+w_y} I^{(i,j)}. \tag{3.3}$$

So, for example, the three block white-black-white feature (type (c)) placed at (x, y), and with block size $w_x \times w_y$ would output the value

$$W^{(x,y,w_x,w_y)}(I) + B^{(x,y+w_y,w_x,w_y)}(I) + W^{(x,y+2w_y,w_x,w_y)}(I).$$

In 2001 Viola and Jones [VJ01] were able to use a combination of these features, strategically positioned on the nose the eyes and so on, to detect human faces. To select these meaningful features among all the possible configurations of position, size, and type, AdaBoost proved to be the key factor. In fact this algorithm can be considered as an aggressive mechanism for selecting a small set of good classification functions which nevertheless have significant variety. Viola and Jones proposed that for each feature, a weak learner determines the optimal threshold (using a simple Decision Stump learner) on the current weighted distribution of the examples, and then just the best features are selected for the current iteration.

The second key idea to obtain an efficient classifier, is to compute the value of the feature in constant time using just few CPU operations, with the help of what they call *integral image representation.*

The integral image at location x, y contains the sum of the pixels above and to the left of the location, inclusive:

$$\Sigma^{(x,y)} = \sum_{i=1}^{x}\sum_{j=1}^{y} I^{(i,j)}$$

where $\Sigma^{(x,y)}$ is the integral image at point x, y as shown in Figure 3.6. It is possible to transform an image into the integral format with just one pass, using the

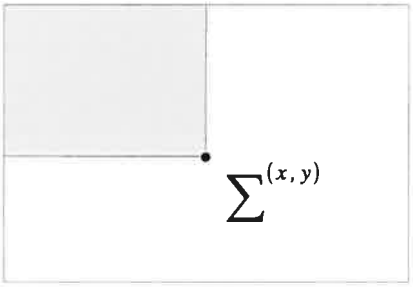


Figure 3.6: The integral image at position x, y is the sum of all the pixels above and on the left.

following pair of recurrences:

$$s^{(x,y)} = s^{(x,y-1)} + I^{(x,y)},$$

$$\Sigma^{(x,y)} = \Sigma^{(x-1,y)} + s^{(x,y)}$$

where $s^{(x,y)}$ is the cumulative row sum, $s^{(x,-1)} = 0$ and $\Sigma^{(-1,y)} = 0$.

Once the integral image is ready, any block B or W can be computed in constant time by using just four references, as showed in Figure 3.7, or more formally with the equation

$$B^{(x,y,w_x,w_y)}(I) = \Sigma^{(x,y)} + \Sigma^{(x+w_x,y+w_y)} - \Sigma^{(x,y+w_y)} - \Sigma^{(x+w_x,y)}. \quad (3.4)$$

Viola and Jones also pointed out that the rectangle sum can be expressed as a dot product, $I \cdot D$, where D is the rectangle image (with value 1 within the rectangle of interest and 0 outside):

$$I \cdot D = \left(\int \int I \right) \cdot D''.$$

The integral image is in fact the double integral of the image (first along rows and then along columns). The second derivative of the rectangle (first in row and then

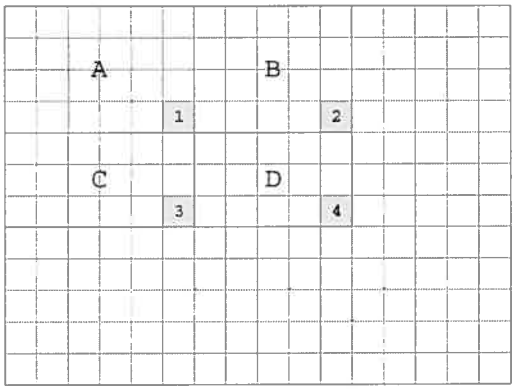


Figure 3.7: The sum of pixels within area D can be computed with just four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D is therefore $4 + 1 - 2 - 3$.

in column) yields four delta functions at the corners of the rectangle. Evaluation of the second dot product is accomplished with four array accesses.

3.4.1 Reducing the training time

One of the problems with the Haar-like features approach is that at each boosting iteration literally hundreds of thousands possible candidates are evaluated to find the optimal set of parameters. This results in a computational cost, for the training process, in the order of weeks on a single Pentium 4 machine. To reduce the computational demand, we have implemented a solution driven by the observation that small changes in the feature's property do not change the overall error significantly. Figure 3.8 shows the error map of the two parameters of the first feature type (a), on the speech/music discrimination problem described in Section 4.2, for the first and 100th iteration. Notice that the two-dimensional space is smoothly changing, especially at the first iteration. This suggests that it is unnecessary to exhaustively sample all parameters combinations.

Hence, we select a random point in the feature space (that is, a set of initial

parameters), and we iteratively adjust until we arrive at a local minimum in a *gradient following* fashion.

As boosting proceeds, AdaBoost concentrates on more difficult examples and the error surface becomes less smooth. This makes the gradient approach less effective, as the local minima is less likely to be close to the optimal. A solution would be to increase the number of sampling parameters, but this would result in a costly search in the early stages of boosting, because the gradient is smoother, and therefore the number of adjustments higher.

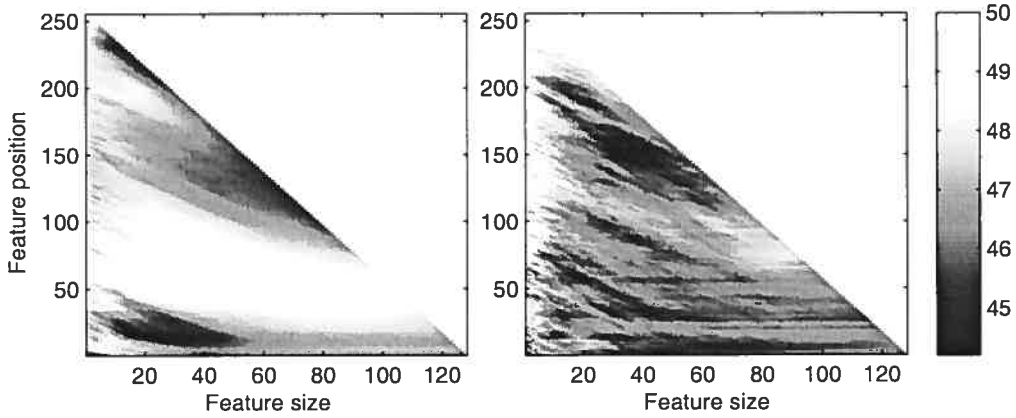


Figure 3.8: The error as a function of two feature's parameters in the first (left) and 100th iteration (right) of the speech/music discrimination problem. Dark areas represents local minima that the algorithm is looking for. The height of the feature decreases linearly with the position, therefore the map is triangular. Observe that the resulting two-dimensional terrain is relatively smooth, suggesting that exhaustive sampling of these parameters may be unnecessary.

We can combine both strategies by setting a time limit for the search of the parameters. On each boosting iteration, we randomly select the first point and adjust the parameters until we reach a local minimum. If we are within the time limit we select another point and we proceed in the same way. This has the advantage of allowing an exploration of a larger space in both an intelligent (with gradient descent) and sparse way (when gradient descent is not effective enough).

The feature thus discovered is suboptimal with respect to the exhaustive search. However, AdaBoost requires only that features support discrimination better than chance. To compensate for the poorer features found, a larger number is required. In our experiments we added in the order of 10-20% more features depending on the problem and on the number of starting random configurations.

One alternative motivation for the suboptimal features selection comes from LazyBoosting by Escudero et al. [EeGR00]. They showed that even with randomly (but homogeneous) features, if the number is large enough it is possible to achieve results comparable to those obtained from the optimal ones.

3.5 Conclusion

In this chapter we discussed several techniques to extract features which captures salient aspects of the music. The quality of a musical sound, that distinguishes different types of sounds are captured by timbral features. We have seen eight standard features of this type in Section 3.1, and we showed a possible approach in Section 3.2 to reduce the dimensionality of these features, by segmenting the song into partitions.

The temporal structure of the music is another important feature, which is captured by autocorrelation and phase autocorrelation, discussed in Section 3.3.

Finally, in Section 3.4 we have proposed a novel approach to capture patterns inside visual representation of the music, using a robust object recognition algorithm.

It is interesting to notice that the acoustic-feature extraction works in fact as a heuristic dimensionality reduction of raw audio data. One imagines that a careful treatment of state-of-the-art dimensionality reduction algorithm to this domain would yield improvements.

CHAPTER 4

EXPERIMENTS

In this chapter we expose the results of the experiments on the algorithms and features that we have seen in the previous chapters, and we show how they can be combined to build effective classifiers. In Section 4.1 we evaluate the hyper-parameters of AdaBoost.MH, the weak learners, the Haar-like features, and a subset selection method to speed up learning.

We used the Haar-like approach to build a robust and fast feature extractor for music information retrieval, and we applied it to the task of speech/music discrimination on audio signal. Section 4.2 describes the approach and the results we obtained.

In Section 4.3 we combine the features discussed in Section 3.1 into the AdaBoost.MH framework to build a state-of-the-art music genre classification algorithm. In the section that follows (4.4) we describe our performance at the recent MIREX 2005 international contest in music information extraction, where our approach ranked first in audio genre classification among 15 submissions, and the second-best among 10 submissions at recognizing artists.

4.1 Experiments on the MNIST Database

The MNIST is a database of handwritten digits [LBBH98], created by LeCun et al. 1998. It is a subset of a larger set available from NIST, and with its total amount of 70,000 examples (60k for training and 10k for testing) is a very good database to evaluate and compare algorithms and pattern recognition methods on real-world data. MNIST has several other interesting features:

- The digits have been size-normalized and centered in a fixed-size image of 28 by 28 pixels.
- It has been used by a large number of algorithms, making it the perfect

testbed for comparison.

- It represents images, which makes it ideal to test Haar-like features.
- The data is ready to analyze in a friendly format.

We used MNIST to evaluate the differences among the weak learners, and to estimate the optimal hyper-parameters of our methods. All the following experiments, with the exclusion of the comparison with other algorithms in Section 4.1.6, are done a training and validation datasets created by splitting the original training set with a ratio of 80%-20%. This is done to avoid the bias on the evaluation of the final dataset. K-fold separation was not possible due to the large size of the dataset and the considerable amount of time required for training.

While the number of iterations of boosting is technically a hyper-parameter of AdaBoost, we limited the iterations to 10^4, where in all our experiments the test error showed a relative stable plateau. In fact, the choice for the optimal number of features does not have the same importance as in other machine learning algorithms (e.g., neural networks) because of the intrinsic resistance of AdaBoost to overfitting: even if the training error tends to zero, the test set error does not increase. It is therefore less important to find a specific stopping point, except for efficiency reasons.

4.1.1 Raw Data Experiments

In the first round of tests we compare the three variants of weak learners, AB.Single, AB.Multi and AB.Tree, on the *raw data*. Because the training time of the tree-based weak learners is extremely long compared to stump-based weak learners, we ran our experiments only on the faster, single-threshold version of the tree.

Each dimension of the input data represents a pixel of the image, and it is considered independent from the others.

Figure 4.1 shows the training and testing curves for the three weak learners. AB.Multi has a steeper gradient both on training and test, suggesting that it can

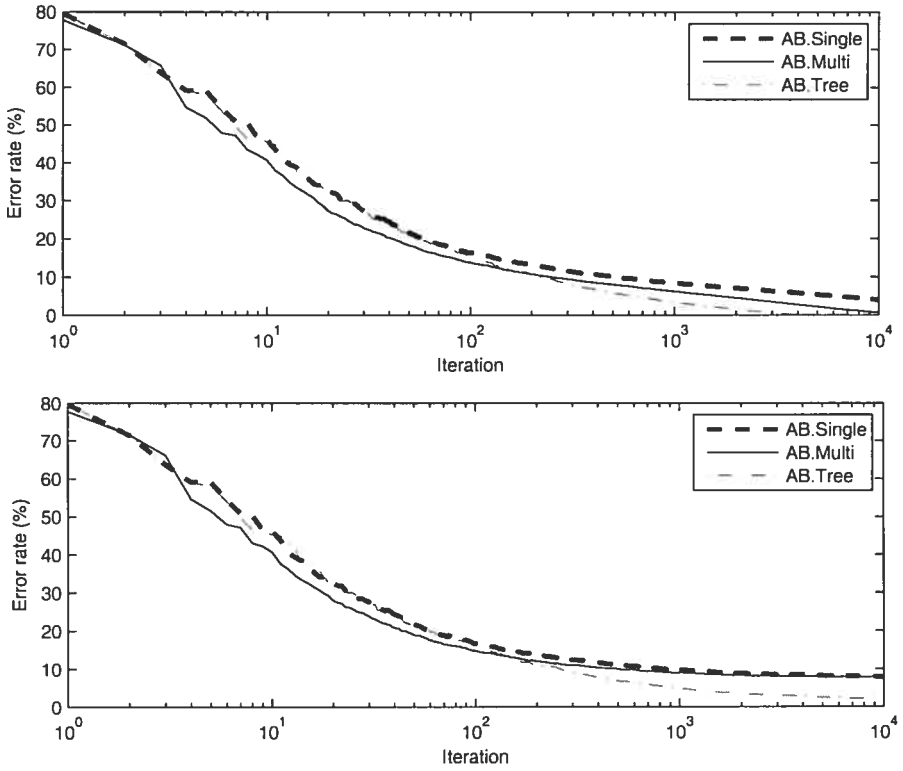


Figure 4.1: The training (top) and test (bottom) curves, on the raw pixel data. The tree-based algorithm AB.TREE clearly outperforms the others because it can build relationships among the pixels of the image.

capture the underlying pattern faster than AB.SINGLE, thanks to the richer base classifier. In fact, results similar to 10^4 iterations of AB.SINGLE are reached by AB.MULTI at about 2000 iterations. Given that during the testing phase, its big-O complexity is comparable on the two algorithms, AB.MULTI is a better candidate for a usable classifier. This trend will be visible on all the tests that we will carry out with these two stumps-based algorithms.

It is important to remember that the curves of AB.SINGLE and AB.TREE, are likely to be similar in the first iterations, because the tree-building strategy described in Section 2.3 uses previously found features to build the model.

Weak Learner	Test error	Training time (hours)
AB.SINGLE	7.85 %	79
AB.MULTI	7.78 %	102
AB.TREE	2.16 %	834

Table 4.1: Accuracy and training time for the raw MNIST data, using three different weak learners within ADABOOST.MH.

The results of Table 4.1 clearly show that AB.TREE outperforms the other weak learners. This is the best proof that the original assumption, that each point is independent, is clearly false. Unfortunately, the advantage of the tree approach is paid with the 8-fold increase in the computational time. All the values showed in this table are referring to the test error after 10^4 iterations, and on a AMD 64 (3200+) machine.

4.1.2 Haar-Like Features

Since the previous assumption that each pixel is an independent entity is incorrect, we need a method that can take into consideration the dependencies among them. AB.TREE showed this ability, but evaluating every possible combination of pixels can be very expensive.

By using the Haar-like features, described in Section 3.4, we can avoid the problem in a simple way. Each feature computes the difference among two groups of pixels, and returns a scalar. This number represents the example in the space of the feature where the weak learner will find the optimal threshold.

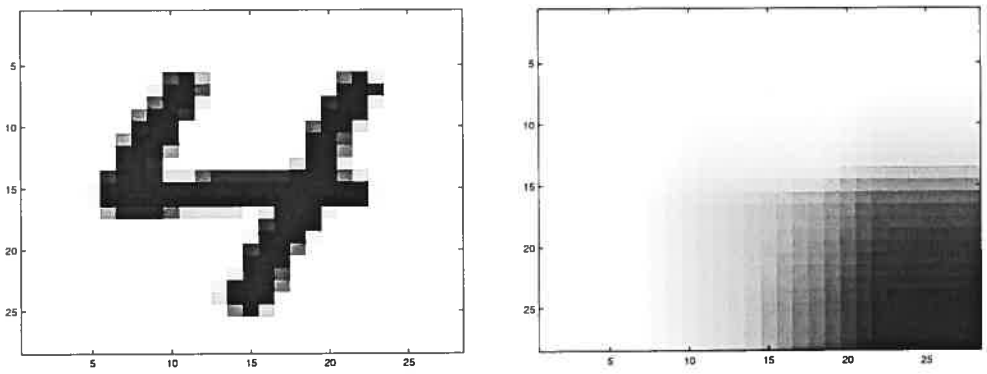
In order to build a good binary discriminator, we must first find a set of salient features that separate the two classes with the largest margin possible. Viola and Jones, in their face recognition algorithm [VJ01], positioned the Haar-like features in different key areas of the picture, such as the nose and the eyes. For the classification of digits, these features are likely to be positioned in areas that can distinguish among group of digits or single digits against the others. The parameters of these features are obtained using ADABOOST.MH, which selects the features that yield the lowest weighted error.

Formally, each Haar-like feature $g_j(I_i)$ returns a real number for each image I_i in the training set, which is the jth attribute $x_i^{(j)}$ in the observation vector $\mathbf{x}_i$, that is

$$x_i^{(j)} = g_j(I_i).$$

Then for each feature j, the best decision stump is found over all the projections $x_1^{(j)}, \cdots, x_n^{(j)}$. Finally, we select the weak learner $h^{(t)}$ which minimizes the weighted training error of Equation 2.1 among all the candidates features.

The computation of each feature can be done in constant time as the image is pre-processed with the *integral image representation.* This cost of this operation is $O(j)$ with j as the number of pixels ($28 \times 28 = 784$ in the MNIST case), and it is generally performed before the training procedure, thus only once. Figure 4.2 shows an entry of the MNIST database in its original format, and processed as integral image. The original structure is still visible in the transformed representation, but now each block can be computed with just four basic CPU operations.



(a) Original Image (b) Integral Image

Figure 4.2: An image representing the number "four" extracted from the MNIST database, and its integral representation. Note how the pattern of the original digit is still visible in the processed integral image.

In the following tests we will be using all the four types of features described in Figure 3.5 of Section 3.4. For each of them we applied the discrete gradient

approach with the time limit fixed to 2 minutes for each feature in each iteration.

Similarly to what we have observed in Section 3.4.1, the trend of the error surface becoming "rougher" is clearly noticeable, as in the first 100 iterations we have an average of five configurations being explored before the local minimum is reached, where the number of parameter's adjustments are in the order of tens. At the 1000th iteration the number of starting configurations evaluated within the time limit has risen to 15, with an average of 5 adjustments each.

The results, summarized in Figure 4.3 and Table 4.2, show a clear improvement in the overall performance, in comparison to the raw pixel learning, on every weak learner.

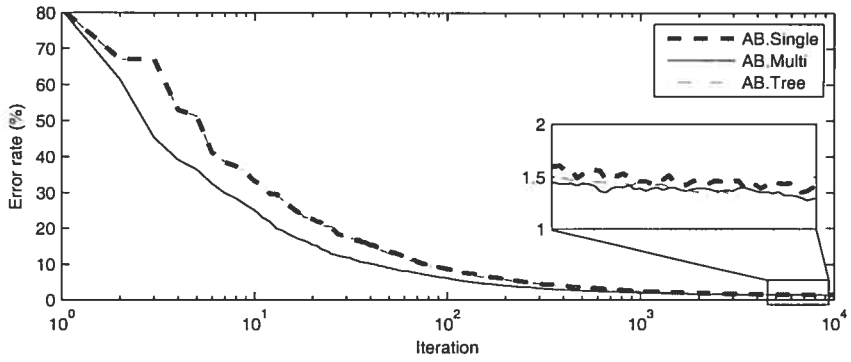


Figure 4.3: The test error using the Haar-like features. The curve of AB.TREE is incomplete as the algorithm did not compute all the iterations within the time limit of 100 days.

Now that the pixels are grouped together, the advantage of AB.TREE over the other weak learners is much less evident. Similarly to what we had for the raw data, AB.MULTI shows a steeper curve, reaching the plateau many iterations before the others.

Considering that the total time per iteration is a fixed parameter, the training time is very similar for the stump-based weak learners, that is AB.SINGLE and AB.MULTI. The construction and evaluation of the tree is done by the algorithm *after* all the other four types of features have been evaluated, but is still bounded by

Weak Learner	Test error	Training Time (days)
AB.SINGLE	1.34 %	72
AB.MULTI	1.31 %	74
AB.TREE	1.33 %	> 100

Table 4.2: Accuracy and training time using the Haar-like features.

the time limit of two minutes. This explains the additional training time observed for AB.TREE, which roughly correspond to the cost of the supplementary step. However, we were not able to compute all the 10^4 iterations for AB.TREE within the time limit of 100 days.

The large amount of training time is balanced by the speed of the model, once it is built. To evaluate the 12k examples of the test set, the algorithm takes 95 seconds for the stumps-based weak learners, and 263 seconds for the tree.

In the following experiments, in which we evaluate the hyper-parameters of the model, we selected as comparison the fastest weak learner, namely AB.SINGLE with the described parameters.

4.1.3 Regularization

As we have seen in Section 2.4.2, regularization can help to improve the test error in some particular conditions. If the positive penalty can reduce the impact of overfitting, a negative penalty, interestingly, can improve the outcome when we are overfitting. With large negative values, ADABOOST is comparable to bagging as the weights of base classifiers become more uniform.

All tests in the previous section have been carried with a value of the regularization parameter θ, equal to zero. We evaluated the same dataset with the same hyper-parameters, changing only θ. The results are summarized in Figure 4.4.

Contrary to general experience, slightly increasing θ did not improve the performance. Please note that in one case, with $\theta = +0.01$, the algorithm stopped prematurely at about 2/3 of the total number of iterations computed by the other configurations, as condition (2.23) was not true anymore.

Also having negative θ did not help at all, and large negative values did signif-

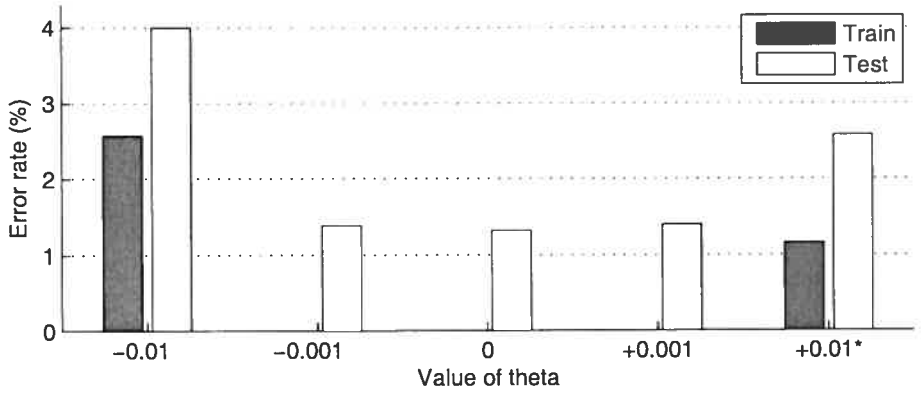


Figure 4.4: The evolution of slightly changes in θ. Note that with $\theta = +0.01$, the algorithm was forced to stop earlier.

icantly decrease the performance.

4.1.4 Abstention

	Test error
With abstention	1.34 %
Without abstention	1.41 %

Table 4.3: The results with and without abstention on the classes, using AB.SINGLE.

Abstention did not show any particular increase in performance. A possible explanation for this is that no class was easier than others to learn.

4.1.5 Feature Selection on a Subset of Data

Even by using the discrete gradient approach showed in Section 3.4.1, the number of features that must be evaluated at each iteration is still very large. This problem is amplified in presence of a large number of examples of the training set, because of the $O(n \log(n))$ complexity needed for each feature. On the MNIST database, with its 60k examples dataset, this is an important factor. To solve this

problem a possible approach is to select the feature using just a subset of the examples, selected randomly at each boosting iteration. Given that the size of the subset is fixed to 10% of the original, we reduced the training time per iteration to less than 1/10th of the original, that is 12 seconds. This reduction is not proportional to the one of the space (as the complexity is $O(n \log(n))$), therefore we have more time per iteration which should compensate the loss in richness of the input.

	Test error	**Training time (days)**
Baseline	1.34 %	72
Subset	3.40 %	9

Table 4.4: Results by taking a subset of the data at each iteration.

Table 4.4 shows the outcomes of the subset approach compared to the standard approach. Taking a random sample of the data helped in terms of training time, but we also observe a substantial degradation of the performance. Unfortunately we did not have the time to evaluate all the aspects of this approach, for instance the size of the subset or an "intelligent" selection of the examples based on the weight, such that the "harder" examples would be chosen with higher probability. An interesting element that supports this idea is the fact that the training error is still far from zero and apparently decreasing, after the 10^4 iterations. This likely means that the algorithm is underfitting, as most of the chosen examples do not help the learning process.

4.1.6 Comparison with Other Algorithms

As said in the introduction, the MNIST database has been evaluated with a large number of algorithms. This allows us to compare our approach with the current state-of-the art in character recognition.

Table 4.5 compares the results of our algorithm, using AB.SINGLE, with Support Vector Machines (SVM) and Artificial Neural Networks (ANN) on a separate 10k examples testing set, after having re-trained our model with the full training database of 60k examples. For simplicity we omitted the results based on pre-

processing and those based on algorithms we did not discuss. Nevertheless, the current winner, convolutional networks with cross-entropy, is shown for comparison. The full table of results is available on the MNIST webpage.

Algorithm	Test error
2-layer NN, 1000 hidden units	4.5 %
3-layer NN, 500+150 hidden units	2.9 %
2-layer NN, 800 HU, cross-entropy [elastic distortions]	0.7 %
SVM, Gaussian Kernel	1.4 %
Convolutional net, cross-entropy [elastic distortions]	0.4 %
Our approach (AB.SINGLE)	1.3 %

Table 4.5: Results of the Haar-like approach compared to other algorithms on the MNIST database. Elastinc distortion is a technique to increase the number of examples by slightly modifying the shape of the input.

4.1.7 Discussion

The Haar-like approach has proved very effective in reducing the test error on data which shows high correlation among the single dimensions, such as for the digits of the MNIST database. Even if simple, these features are capable to capture complex patterns. Figure 4.5 shows the reconstruction of the ten digits using the features found during the training.

The images are created by evaluating all the examples of the training set with each feature. If the feature vote in favor of the class of the example, the pixels covered by the white area of the feature are decreased by the confidence of the feature, and the black pixels are increased. If the feature vote against, we do the opposite. These images represent in fact the learned "average" of the class over all the examples.

Unfortunately, this method is extremely slow during training, a problem already highlighted by Viola and Jones when they first described the algorithm. The discrete gradient descent technique allows us to avoid the costly full search and obtain comparable results, but if the space of the parameters is too large, it might still take weeks to obtain the required set of features. Subsampling does not seem

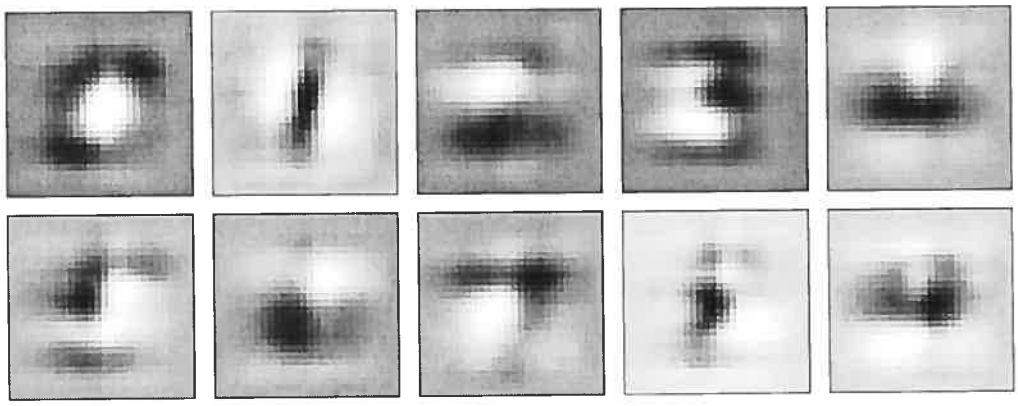


Figure 4.5: The reconstruction of the ten digits using the Haar-like features found during training. Each figure is the output of the feature for a single class, on the examples that belongs to this class. Dark areas represent a predominance of positive vote, where white areas are voting extensively against.

to help, even if it is a promising approach.

We also investigated the possibility of using the "experience" collected with the model to direct the search of the optimal parameters and thus reduce the training time. By taking the 10^4 features previously found, we can compute a map, showed in Figure 4.6, where the coordinates correspond to the height and width parameters of the feature. In each cell we find the summed confidences α of the features that share the same parameters.

The map is used to select the size parameters of a candidate, by using a stratified sampling which takes the most dominant feature with higher probability. The other parameters, that is the coordinates of the feature, are selected at random. Unfortunately, our preliminary results showed that even if the learning curve was steeper, all the advantage was lost after about 1000 iterations, where the plateau of the test error curve is still far away.

Nevertheless, even with such long training times, if the objective is to build a working classifier, the cost for the double training is counterweighted by the speed during detection, because of the smaller number of features that are needed to

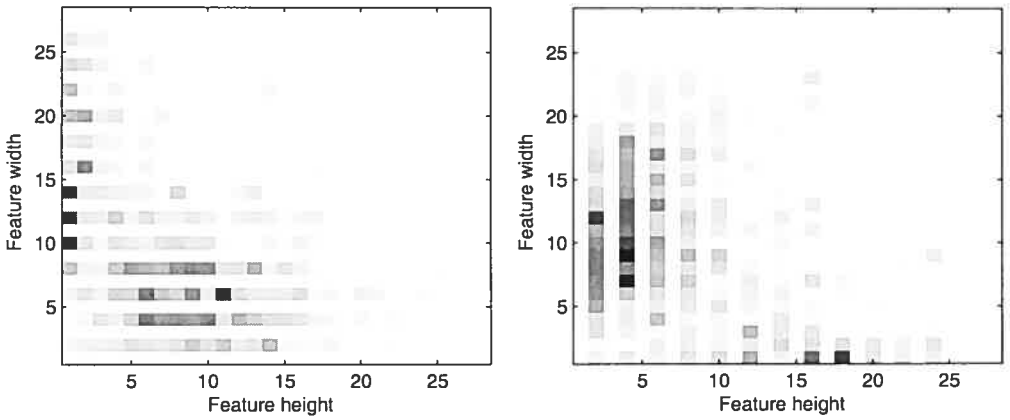


Figure 4.6: The probability map of the first two feature types ((*a*) on the left and (*b*) on the right). Dark cells represents the dominant configurations, that is with larger weight or which have been selected more ofter.

achieve the same precision.

Extensions of the algorithm, such as the regularization and abstention, appear not to be critical in this multi-class settings. The choice of the weak learner does make a difference as long as we are interested in the speed of the final classifier. In that sense AB.MULTI appears to be the best candidate, considering that it can obtain results comparable with the others with a lower number of iteration/features.

4.2 Frame Level Speech/Music Discrimination

The ability to automatically discriminate speech from music in an audio signal has become a topic of interest in the last few years, as it is an important task in many multimedia applications, such as automatic audio news transcription of a radio broadcast, where non-speech parts would presumably be discarded, or low bit-rate audio coding, where multi-mode coders adjust themself depending on the type of signal.

Saunders in 1996 [Sau96] proposed a real-time speech/music discriminator based on a multivariate Gaussian classifier that used four statistical features on the zero-

crossing rate and one energy-related feature. In 1997 Scheirer and Slaney [SS97] extended the number of features to include the evaluation of pitch, amplitude, cepstral values and line spectral frequencies (LSF). More recently, other approaches have used the posterior probability of a frame being in a particular phoneme class [WE99], HMMs that integrate posterior probability features based on entropy and "dynamism" [AMB02], and a mixture of Gaussians on small frames [ER02].

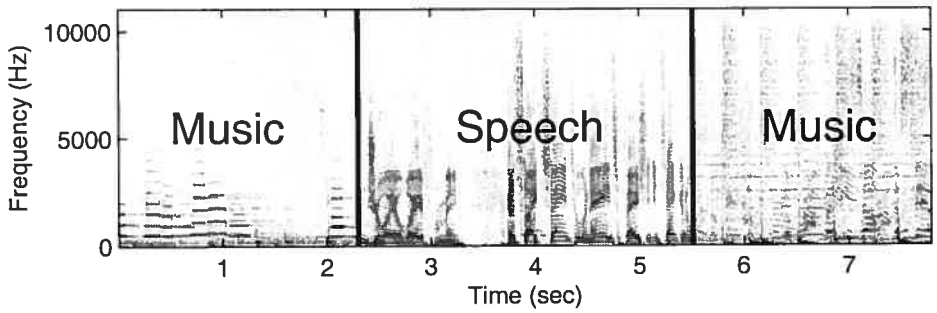


Figure 4.7: The pattern that characterize speech is clearly visible in a common rendering of sound such as the spectrogram.

We have adopted the Haar-like features approach discussed in Section 3.4 and investigated on the MNIST database, to find regular geometric patterns in speech and non-speech audio spectrograms. These regularities are detectable visually, as demonstrated by the ability of trained observers to identify speech structure (e.g. vowel formant structure, consonant onsets) and musical structure (e.g. note onsets and harmonic pitch structure) through visual inspection of a spectrogram such as the one showed in Figure 4.7.

Despite being motivated by work in vision, this model is well suited for audio signal processing. Though it treats individual 20ms slices of music as having fixed geometry, it places no limitations on the geometry of entire songs. For example, it places no constraints on song length nor does it require random access to the audio signal. In other words, this approach is causal and is able to process audio streams on-line and in real time.

The model can be easily extended to incorporate subsequential frames (for instance 50 frames corresponding to a second) as an additional dimension, in order to detect short time varying patterns, such as the decay of an instrument. In this case the representation will become again two-dimensional with the frequency on one axis and the time on the other. Such approach will have to address the sliding of the spectral representation, but this issue can be solved by limiting the number of frames, overlapping them, or more simply with a sliding window over the signal.

4.2.1 The Algorithm

The process of selecting the most salient features is in this case simpler than with digits, because the problem is binary and because we are limited to a single dimension instead of two, as we are just taking the Short Time Fourier Transformation (STFT) of the signal. For the second reason the only features we are using are of type (a) and (c) (see Figure 3.5).

We define each example $\mathbf{F}$ in our training set as a 20ms STFT frame of an audio source, and $F^{(i)}$ as the signal intensity at frequency i. Thanks to the fact that we are just looking for position and size of our feature in the frequency space, we can simplify Equation 3.2 seen in Section 3.4 to

$$B^{(i,w_i)}(\mathbf{F}) = \sum_{j=i}^{i+w_i} F^{(j)},$$

where the black stripe with its upper side is placed at (i) with size w_i. Similarly Equation 3.3 becomes

$$W^{(i,w_i)}(\mathbf{F}) = -B^{(i,w_i)}(\mathbf{F}) = -\sum_{j=i}^{i+w_i} F^{(j)}.$$

The computation of the integral "image" (in the monodimensional space, it is actually the integral function), is now a very simple additive sum of the STFT case, as Figure 4.8 illustrate. This allows us to evaluate a very large number of

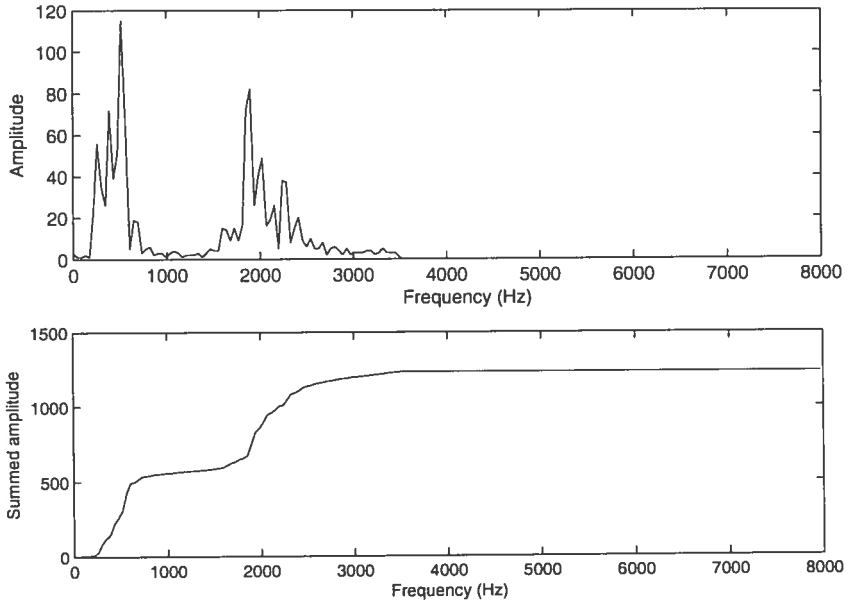


Figure 4.8: The integral "image" (bottom), of a STFT (top) is a simple additive sum. The amount of energy within any range can be found in constant time, with just a subtraction of two index point in the integral image.

candidate features in every boosting iteration as the space is much smaller than it was with digits.

The value returned from the feature is used in the classification, and as we did previously with MNIST, we select only the features and the parameters that minimize the weighted error among all the examples. The discrete gradient descent approach of Section 3.4.1 is also used to speedup the training process.

Despite the simplicity of the features, they can discriminate between speech and music by capturing local dependencies in the spectrogram. For example, the three-block feature (type (*c*)) depicted in Figure 4.9 is well-correlated with the speech signal and quasi-independent of the music signal. Figure 4.10 displays the real-valued output of the feature for all training points, and the threshold of the optimal decision stump. This feature alone, selected in the first iteration of AdaBoost,

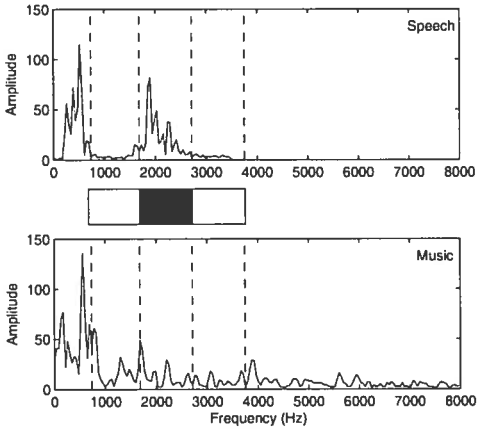


Figure 4.9: This three-block feature can distinguish speech from music. It is well correlated with the speech signal (its output is 347), and independent of the music signal (its output is -577).

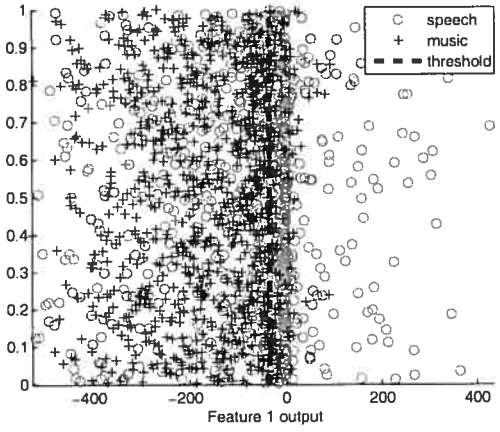


Figure 4.10: The output of the feature showed on the left for all training points, and its decision stump's threshold. The data has been randomly distributed on the vertical axis for clarity.

has already a 30% error rate on the test set.

4.2.2 Experimental Results

In the experiments, we used 240 digital audio files of 15 second radio extracts published by Scheirer and Slaney [SS97][1]. We extracted 11200 20ms frames from this data, normalized them, and then processed them with FFT, RASTA [HMBK92] (a high-pass (or band-pass) filter applied to a log-spectral representation of the signal such as the cepstral coefficients), and log-scale FFT. We chose the first because it is the simplest representation of the frequency spectrum, and the other two because of their popularity in speech processing. The FFT represents most dimensions input (in frequency) with 256 points, followed by the other two (respectively 26 and 86). The size of this space has an important impact on the training time, but it is far away from the one of MNIST which was 784. During detection, again, the size of the space does not play any role, thanks to the integral

[1]The data was collected at random from the radio by Eric Scheirer during his internship at Interval Research Corporation in the summer of 1996 under the supervision of Malcolm Slaney.

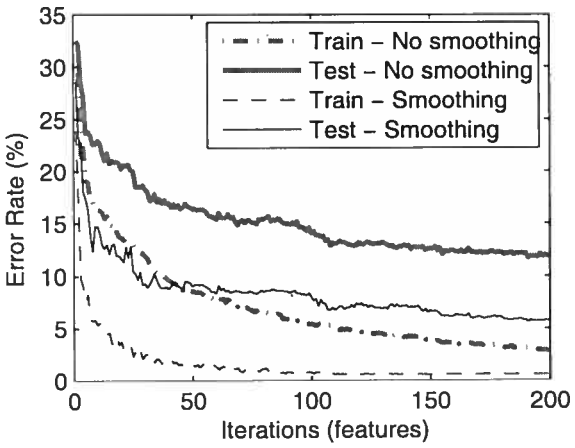


Figure 4.11: The results on a 20ms FFT frame, without and with smoothing. The benefits of smoothing are clearly seen both in test error and train error.

Table 4.6: Testing error rate using single frame features.

	Without smoothing	**With smoothing**
FFT	$11.9 \pm 2.0\%$	$6.7 \pm 1.6\%$
RASTA	$10.8 \pm 2.8\%$	$7.2 \pm 1.7\%$
Log-FFT	$12.6 \pm 2.4\%$	$7.4 \pm 2.1\%$

function representation.

Figure 4.11 shows the training and test errors using the FFT. We can observe that at a frame level, on a simple FFT we already obtain an error rate of about 12% after 200 iterations, which is far better than the 37% of the best frame-level feature of Scheirer and Slaney on the same dataset.

We can improve the classification by considering the previous frames with a simple smoothing function. Let $f(\mathbf{x}_i)$ be the output of the strong learner, where $\mathbf{x}_i$ is a frame at time i. Then, the new output used for classification is

$$g(\mathbf{x}_i) = \frac{\sum_{j=max(i-nframes+1,0)}^{i} \beta^{i-j} f(\mathbf{x}_j)}{\sum_{j=max(i-nframes+1,0)}^{i} \beta^{i-j}}, \tag{4.1}$$

Table 4.7: A comparison of various *frame level* features. The value of the first column represent the CPU usage during real-time discrimination. This value for our approach is estimated.

Feature	**CPU**	**Error**
Rolloff	17%	$46 \pm 3\%$
Spectral Centroid	17%	$39 \pm 8\%$
Spectral Flux	17%	$39 \pm 1\%$
ZCR	0%	$38 \pm 4\%$
Ceps Residual	46%	$37 \pm 7\%$
Our Approach	*1%	$12 \pm 2\%$

where β is a decay parameter between 0 and 1 and where $nframes$ is an integer that corresponds to the number of past frames to consider. In order to find the best values of a and $nframes$, we randomly concatenated the audio (wave) files of the validation set and measured the classification error rates for several values for the decay parameter. We chose this procedure in order to approximately simulate audio streaming from a radio station and get the best values at $\beta = 0.98$ and $nframes = 40$. With these settings the error reaches a value less than 7% with less than a second of information, and converges after about 150 iterations; but even with a much smaller number of features, such as 75, the error level is below 10%. Another interesting solution would be to use HMM's on the classifier's outputs.

RASTA and logarithmic scale FFT representation did not perform as well, even though the results are still below 10%. This is quite surprising, as these preprocessing are widely used in speech recognition and are considered particularly good for speech analysis. However, it is possible that these features discard precious information at higher frequencies, where voice is not present.

Our approach is highly competitive compared to others common *frame level* features. Table 4.7 shows a comparison with Scheirer and Slaney's frame level features. Although they have been using a different classifier (mixture of Gaussians), the large gap between the error rates of these simple features and ours, clearly shows that the predictive power of our learned model is much higher. The CPU

usage refers to real-time discrimination on a 150Mhz Alpha machine, which we unfortunately did not have access to. However, the amount of computation needed for our features is insignificant. Even if all our 200 features where constituted by three blocks (the most complex feature), the total computation would be $3 \times 200 = 600$ simple subtractions, once the integral function has been computed.

4.2.3 A Winamp Plugin

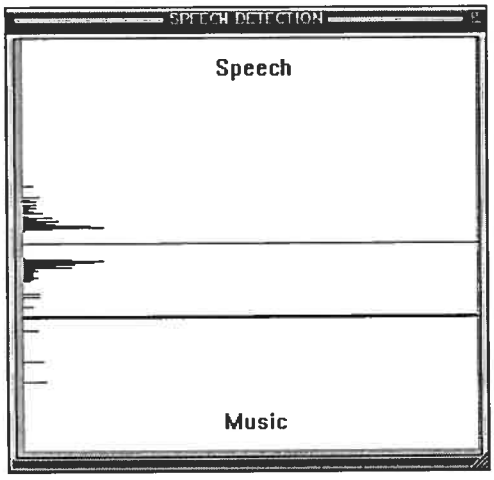


Figure 4.12: A Winamp plugin version of our model. The long thin horizontal line displays the decision boundary between Speech (top) and Music (bottom). The long thick horizontal line displays the decision of the model, with vertical position proportional to confidence. Here, for example, the model is moderately confident that the stream is Music (in this case, a correct prediction). The shorter horizontal lines on the left provide a histogram of the decisions and confidence ratings of the weak learners. See text for details.

Winamp (`www.winamp.com`) from Nullsoft is a popular audio player for the Windows operating system. Like many multimedia software applications, Winamp uses modular "plug ins" that extend the functionality of the basic player. Because the development kit for writing these plugins is available for public use, we were able to develop a version of our model that runs within Winamp. This plugin is able to predict in real time whether the audio stream being played in the Winamp

player contains music or speech.[2] Because the plugin is distributed already trained, it requires few computational resources and should run well on any computer that can run Winamp.

In Figure 4.12 a screenshot of the plugin is shown. The long thick horizontal line displays the decision of the model. The distance of this decision line from the boundary between Speech (top) and Music (bottom) is proportional to the confidence the model has in its decision. This confidence is computed as a sum of weak learner votes (+1 for Speech, -1 for Music) multiplied by the respective weak learner confidences (α, in ADABOOST). See Section 2.2 for details.

The shorter horizontal lines on the left in the screenshot display a histogram of the weak learner votes where the vertical position y of each of these shorter lines is given by the vote (+1 for Speech, -1 for Music) multiplied by confidence. In addition, confidence is also used to scale the width of each of these lines. If two weak learners fall on the same vertical position in the plot, their confidence is summed, generating a single longer line. The three horizontal lines near the bottom of the plot are individual weak learners having particularly high confidence that the stream is music.

The plot is refreshed every 28ms, resulting in a smooth animation of model performance. Though the graphical interface is relatively primitive, it can be instructive to watch model behavior evolve over time in response to an audio stream.

For the Winamp plugin, we trained a version of the classification model using randomly-selected segments from a number of Internet radio streams. Labeling was achieved implicitly by treating talk-radio streams as speech examples and music-radio streams as music examples. Though training streams were not labeled by hand, we did control against contamination of speech with music and vice-versa through careful listening. We did not create a hand-labeled testing set of mixed speech and music radio streams. For this reason we are unable to report a reliable error rate for the Winamp plugin (nor was this our goal). We were able

[2]The plugin is available for free download at `www.iro.umontreal.ca/~casagran/winamp/`.

to demonstrate that the model runs efficiently and perform well in a real-world application. We observed that the output of the model is stable and reliable for a range of input streams.

4.2.4 Discussion

Haar-like features are very effective in exploiting geometric regularities in a fixed-sized two-dimensional representation of frame contents, and can be used to perform frame-level classification of audio.

Because of the strong relationship among frames in time, we can increase the performance of the classifier with a simple smoothing on the output of the frame-level classifier. It would be interesting to use an HMM instead of the smoothing function on the classifier's output, as HMMs are very well suited for such a task. It is also possible to do training directly on a set of subsequent frames to capture local dependencies in the time domain. However, such an approach would have to deal with the problem of the absolute position of the features in time, and therefore would probably be working only if the number of frames is limited.

To make an example of this problem, one can imagine a bloc of frames of one second (50 frames) over a 5 minutes' song. At time 2:20 the frames enclose the sound of a trumpet. But just after half a second, the trumpet is not anymore enclosed within the bloc, and cannot be used as a reliable training example as it does not represent a meaningful pattern.

To overcome this problem the frames can be overlapped, but this would introduce noise which makes the classification harder. A better solution would be to train *windows* of the sound that we would like to map, and then "slide" a *detector* with the same size as the window over time, and measure the vote and confidence at each step. That is the solution used by Viola and Jones in their previously cited paper. Each of their examples was a 24×24 picture of either a face or something else. In the large image where they are looking for faces, the resulting detector is evaluated on every possible position and size (with some δ, as shifting one or two pixels is not significant for the detection). The positive outcome of the detector is

considered a "hit".

This approach is particularly interesting for the task of recognizing instruments in music. Detecting an instrument with just few frames is very hard, even for humans. With the window approach it is possible to catch both temporal and spectral patterns of the instrument itself. The biggest problem would be to create a large enough database of windows in which the instrument is either played or not.

Another interesting aspect that could be explored is the use of different basic features (such as Gaussians or band-passes), and different representations such as wavelets or sine-wave replicas [LBPR01]. Naturally, the extremely cheap computational cost of, respectively, the Haar-like features and the STFT would be lost; but it is also true that for sound there is no "natural" representation, and other techniques might better match the task.

Finally, while the current model is limited to two-class categorization, it is easily expandable to the multi-class case, as we did for digit recognition in Section 4.1, and for music genre recognition in the following section. This would allow us to extend our work to more challenging classification problems such as speaker identification, singer identification and mood identification.

4.3 Music Genre Recognition

Musical genres are labels given by people to classify the vast universe of music that mankind has created in its history. There is no clear mathematical definition of what a genre is, and labeling can diverge among different cultures, backgrounds or *zeitgeist*. However, if we take a group of people and train them on different genres, they will agree on most labels as demonstrated by many studies [PG99, LTMM04, MAL05]. This suggests that even an automatic system should be able, by extracting auditory features, to classify musical genres.

4.3.1 Haar-Like Features

We evaluated the Haar-like features for the genre recognition task, using the spectrogram and the MFCC representation of sound as the basis for the training data. Instead of taking just a single frame, as we did for the speech/music discrimination task, we extended the space of search for the Haar-like features to m frames. The value of m must be a tradeoff between the increased amount of information captured as the number of frames grow, and the time shifting problem (seen in Section 4.2.4), that is the non-fixed location of the pattern within the window which moves as the time goes on.

After evaluating the outcome on a out-of-sample data set we chose to fix m to 10, which corresponds to about 0.2 seconds. Experiments showed an accuracy rate of 33.8% on STFT representation and 44.7% on MFCC, which is similar to the results of Perrot and Gjerdigen [PG99], who demonstrated that humans barely go above 50% of accuracy rate when listening at only 250ms of sound. Unfortunately, as we will see in the following sections, other and much simpler features, can achieve a much better accuracy rate in a fraction of time.

The poor performance of the Haar-like features can be explained by the difficulty of finding a "visual" pattern that characterizes genre, in contrast to voice which has a distinctive pattern. However, this approach might still be a viable solution, if used as "dynamic feature" within a multi features meta-algorithm. For instance the model could be trained to detect a particular instrument, and used to generate a confidence on the presence of that particular instrument. In their winning algorithm for symbolic genre recognition at MIREX 2005, McKay and Fujinaga showed [MF05] that instruments are a key factor in the identification of genre. Unfortunately these types of experiments require a large dataset of tagged examples which we did not have access to.

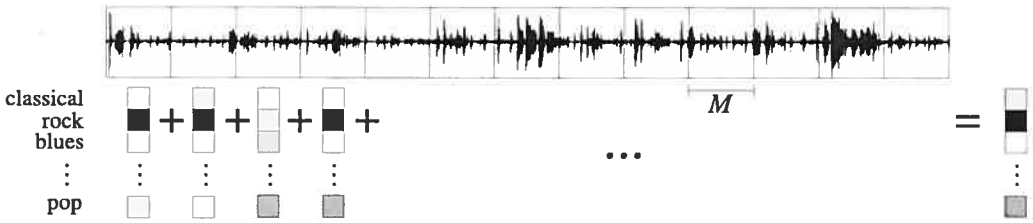


Figure 4.13: The segmentation of the the song $\mathbf{s}$ of length l into $\lfloor l/M \rfloor$ segments of size M. For each segment the classifier returns class posteriors which are summed. The label is selected with a majority vote among the classes.

4.3.2 The Segmentation Algorithm

Given that the naive Haar-like features approach did not work well, we concentrated on the auditory features discussed in Chapter 3 to build our genre recognition algorithm. However, as we have said in Section 3.2, using all the frame level data extracted by the features individually results in observation vectors too large for an effective classification. Methods that aggregate these features over the entire song are promising, though they risk discarding too much in the way of long-timescale temporal dynamics. Models tuned to capture medium to long timescale structures are also promising although they tell us little about fine-grained structure like timbre. Our approach is to treat the size of the aggregate feature as a hyperparameter and optimize it with respect to our classification algorithm.

Briefly, our method is to partition a song into pieces of a fixed size, to classify each piece by AdaBoost.MH, and to label the original song by a majority vote, as shown in Figure 4.13. In more detail, we first choose this fixed size, M, and then partition the song signal $\mathbf{s}$ into $\lfloor l/M \rfloor$ *segments* of length M. The next step is meta-feature extraction. We introduce here the notion of a meta-feature extractor $a_r : \mathbb{R}^M \rightarrow \mathbb{R}^N$ that maps a music signal of length M to a feature vector by using another feature-extractor $r : \mathbb{R}^m \rightarrow \mathbb{R}^n$ (with $M > m$) internally. We compute the meta-features of each segment: $f(s, qM; a_r)$ for $q \in [0, \lfloor l/M \rfloor]$. We classify each of these meta-features independently with a classifier that is both appropriate to the meta-feature, and capable of generating class posteriors. Then we average the

classifier outputs across the set of meta-features to get a class posterior for the song, and assign the label with highest probability.

In this work, we experimented with different classifiers and feature-extractors r, but limited ourselves to the meta-feature extractor a_r whose nature was to compute the sample-mean and sample-variance in each dimension of r's output. This way, our results can be directly compared with West and Cox [WC05], and Tzanetakis et al. [TEC01].

This method generalizes some previous song classification methods. For instance, the method of Xu et al. [XCST03] is reproduced by choosing $M = 1$ (and then ignoring the null variance). The method of [TEC01] is reproduced by setting $M = l$ so that the whole song is captured by a single feature $f(s, 0; a_r)$. One of the methods presented in West and Cox [WC05] is reproduced by setting M to be the number of samples per second of audio. In contrast with [WC05] our method involves fixing M before seeing s. Our experiments are designed to show that at least for some popular features and classification algorithms, the choice of M is relatively easy.

Under the assumption that temporally proximate features tend to be more correlated, this method captures the non-Gaussian distribution of features over time without incurring the cost of fitting a parametric density model such as a mixture of Gaussians. This method also produces larger training sets so that if shorter segments are informative, we can reap the benefits of more effective statistical models.

We performed a series of trials on a genre classification database to explore the sensitivity of classification rates to the choice of the aggregation-segment length M, and to the choice of a classifier. We investigated both the classification error rate of individual segment meta-features and of whole songs after voting, using a variety of frame-level features and meta-feature classifiers.

4.3.3 Data Set

For these tests we used the genre database from the lab of Tzanetakis used in [TEC02], [TC02], and [LT03a]. This database has a total of 1098 songs excerpts. Each song is labeled as one of ten genres: blues, classical, country, disco, hiphop, jazz, metal, pop, reggae or rock. There are more than one hundred examples of each genre, but the habit is to use the first one hundred examples of each label and to ignore the rest. Each example is a thirty-second excerpt from the interior of a longer song, stored as a mono-channel `.au` file at 22050Hz. To our ears, the examples are well-labeled, but exhibit a surprising range of styles and instrumentation within each genre.

4.3.4 Acoustic Features

We investigated the results for several conventional acoustic frame-level feature sets:

- 64 MFCC
- 128 RCEPS
- 1 ZCR + 1 Spec. Centroid + 1 Spec. Spread + 16 Rolloff = 19 MISC
- 128 FFTC

These features were extracted by using a freely downloadable **C** program and complementary library called MSL (Montreal Scientific Library) [Ber05]. Input files were partitioned into contiguous, non-overlapping frames of 1024 samples, so that each frame corresponded to about 47 milliseconds of audio. Thus the first frequency band of the FFTC was about 21Hz. This contrasts with the standard choice of 512 samples (23 milliseconds); we chose the longer arrays because it provided slightly better classification rates in initial comparisons, and we felt that the more refined spectral information might be useful. Recall that in the computation of RCEPS and MFCC features, we must add a small positive constant to each Fourier magnitude

to ensure that we do not attempt to take the logarithm of 0. For our computations we added $\gamma = \frac{1}{5000}$, but we found that classification was unaffected by $\gamma < \frac{1}{100}$.

Feature extraction was quick using this library. On an AMD 64 Processor (3200+) with the database stored on disk, a feature-extraction of 128 RCEPS, 32 MFCC, 16 rolloff, zero-crossing rate, spectral centroid, spectral spread, and 32 LPC for each 47ms frame of the 1000 songs in the database took 3 minutes and 7 seconds.

4.3.5 Classifiers

We used three machine-learning algorithms to classify the meta-feature vectors: support vector machines (SVM), sigmoidal neural network (ANN), and AdaBoost.MH.

Our SVM classifiers were fit using the SPIDER Machine Learning toolbox by [WEBS05], which, in this case, was simply a MATLAB wrapper for SVMTorch by [CB01]. We used a standard Gaussian kernel. We optimized the width of the kernel first—using a relatively low value of the soft-margin parameter—and then optimized the soft-margin parameter C, while fixing the optimal width. Both of the optimizations were done on a validation set and the optimal set of hyper-parameters was selected by 5-fold cross-validation.

The ANN was a feed-forward network of 64 hidden nodes, fit by gradient descent. An implicit $L2$-norm weight-decay was introduced by initializing the network parameters to small values, and early-stopping using a validation set. No explicit regularization was applied during optimization. Classification was performed by bagged ensemble of 10 networks, as described by [Bre96]. The ANNs were simulated using the MSL [Ber05].

Our AdaBoost.MH simulations were done using the algorithm described in Section 2.5.2. In our experiments, overfitting never occurred, and the number of iterations was constrained by training time rather than degradation of the classification rate.

4.3.6 Results

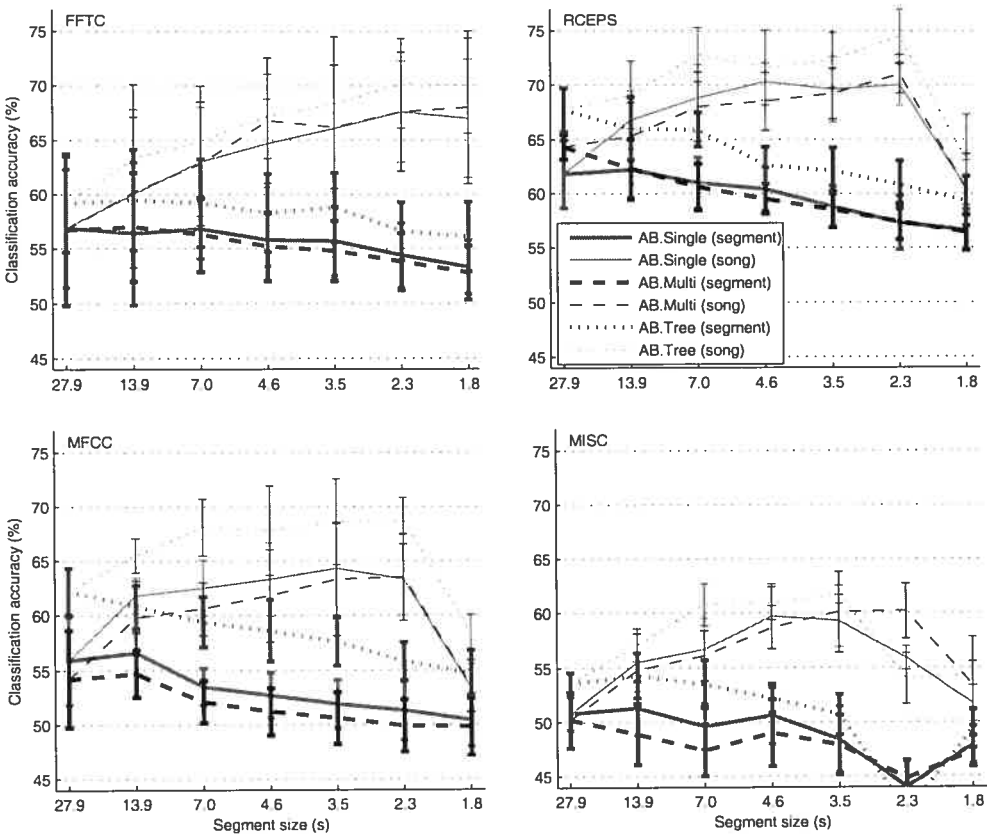


Figure 4.14: The classification accuracy on four different feature sets: FFTC, RCEPS, MFCC and MISC trained with AdaBoost.MH showing the trend with different segment sizes.

Our first step was to evaluate the efficiency of the segmentation approach. We ran experiments on FFTC, RCEPS, MFCC, and MISC on a range of segments between 1.8 and 27 seconds. This latter roughly corresponds to the whole song. All tests were done using a 5-fold cross validation. For all tests, care was taken to split the dataset at the song level, to ensure that the segments from a single song were not split between train and test. However, within the optimization routines that train SVM and ANN models, a validation set is removed from the training data.

At this subroutine level, song membership was not available, and segments were split at random. The consequence was that early stopping and hyper-parameter optimization were biased.

Figure 4.14 shows the outcomes of the two variants of AdaBoost.MH, trained for 2500 iterations each. As segment size decreases we clearly observe differences between the song and the segment classification rates. The superiority of using relatively short segments is seen when these segments are aggregated: they generally outperform the song level features, reaching a plateau at around a segment size of three seconds. We also observe a general degradation of classification rate when using extremely short segments. This can be explained by the fact that the short segments (1.8 seconds) might not be rich enough to capture the information necessary for genre classification.

Similar trends are observed in Figure 4.15, where ANNs and SVMs are compared on the same set of features. Also, Figure 4.16 summarizes the results for every feature and algorithm on an entire song and on a segment size of 3.5 seconds. These preliminary results already give us a good idea of the classification power of each feature. It is clear that each individual feature is strong enough to account for a significant amount of variance. However no individual feature, regardless of segment size, is descriptive enough to drive a competitive classifier.

4.3.7 Discussion

The common trend across these trials is that for each frame-level feature and classifier, the classification rate achieved by the classifier is sensitive to the segment length. In every case, as the segment length decreased from the length of the file to around 3 or 4 seconds, the segment classification rate dropped, while the song classification rate rose. One explanation for this phenomenon is that each segment supported a noisy estimate of the true class posterior; the process of averaging the predictions has the effect of reducing the variance in the estimate. Of course, each of the segments is from the same song, so it is unlikely that the classifier makes an unbiased classification, so that variance-reduction can only provide a limited

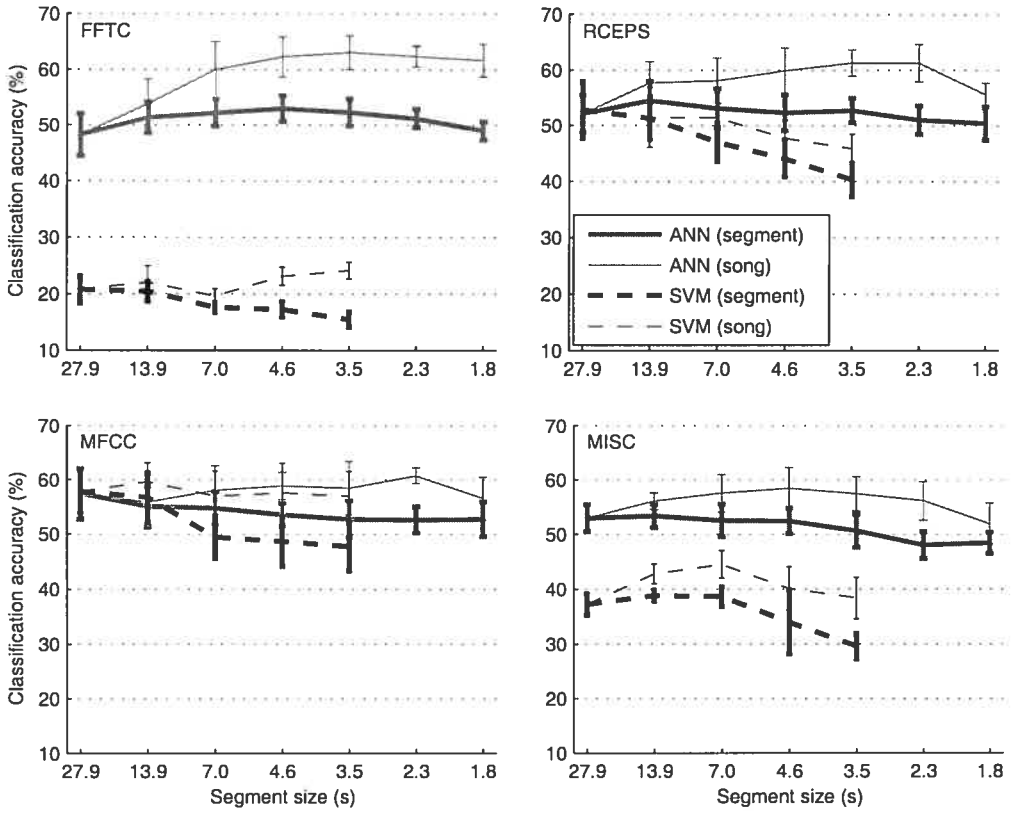


Figure 4.15: The classification accuracy of ANNs and SVMs for different segment lengths. The four curves represent performance on the feature sets: RCEPS, MFCC, MISC and FFTC.

improvement. At the same time, as segments get shorter, the meta-features may become more varied themselves, and make classes less easily separable.

While different (features,classifier) pairs exhibited different behavior on the shortest segment, they ranged from a negligible improvement over 3-second segments, to less-negligible deterioration. We may conclude that the extra computational time associated with the large number of meta-features that result from tiny segments is not warranted, especially in the case of the SVM, whose training time is quadratic in the number of examples.

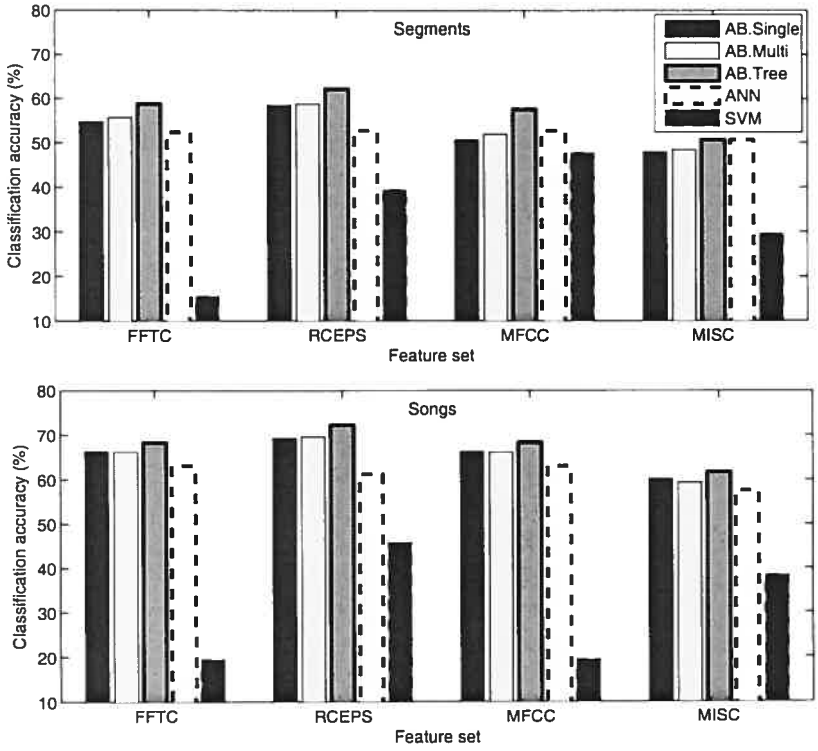


Figure 4.16: The classification accuracy on four different type of features, and all the algorithms. For simplicity we display only the 3.5 second segments.

Another explanation for the increased performance at smaller segment lengths is data amplification. By using shorter segments, we obtain a proportional increase in the number of training examples for each class. Our current training technique does not account for the fact that these additional samples are not statistically independent because they come from the same song. Nevertheless, the increased training-set size may account for better performance.

With respect to the classifiers used, it is clear that for song classification, AdaBoost is a more effective algorithm, compared with a neural network fit by gradient descent and an SVM. AdaBoost is especially efficient at making use of the enormous number of features that can be computed from acoustic data. Ad-

aBoost can also obtain good results in a short amount of time, in contrast to SVMs.

The poor performance of SVM is quite surprising, considered that the MIREX entry from Mandel and Ellis [ME05b] showed a very effective implementation of SVMs, on both genre and artist contests. However, even if they also compute the mean and variance in frame-level features over a larger segment length, as we do, they use these to compute a different kernel. While we used the kernel

$$K((\mathbf{m_1}, \mathbf{v_1}), (\mathbf{m_2}, \mathbf{v_2})) = e^{-\alpha(((\mathbf{m_1},\mathbf{v_1}))-((\mathbf{m_2},\mathbf{v_2})))'\Sigma^{-1}(((\mathbf{m_1},\mathbf{v_1}))-((\mathbf{m_2},\mathbf{v_2})))}$$

where Σ is the sample covariance matrix of meta-features $(\mathbf{m}, \mathbf{v})$, they used the kernel

$$K((\mathbf{m_1}, \mathbf{v_1}), (\mathbf{m_2}, \mathbf{v_2})) = e^{-\alpha KL_s(G(\mathbf{m_1},\mathbf{v_1}),G(\mathbf{m_2},\mathbf{v_2}))}$$

where $G(\mathbf{m_1}, \mathbf{v_1})$ is a Gaussian with mean $\mathbf{m}$ and a diagonal covariance matrix whose diagonal is $\mathbf{v}$. We believe that the kernel they used is a more natural measurement of similarity between meta-features.

4.4 MIREX

MIREX (Music Information Retrieval Evaluation eXchange) is an annual series of contests whose main goal is to present and compare state-of-the-art algorithms from the music information retrieval community. It is being organized in parallel with the ISMIR conference [CS05]. We participated in two contests: Audio Genre Classification and Audio Artist Identification. Since the MIREX protocol asks for an algorithm (model-generator) from each team, and not a single model, we submitted the same algorithm to participate in both contests. This section presents the performance of our contest entry on Tzanetakis' genre database, and summarizes the contest results.

4.4.1 Simulation on Tzanetakis Genre Database

To compete in the MIREX competition we wanted to make a more performant classifier according to what we learned above. We learned that AdaBoost was the more effective classifier for this task, so we wanted to evaluate it on the concatenation of all our feature sets. When we submitted our algorithm, we believed that the following feature sets could be concatenated to provide the most informative input to AdaBoost:

1. 256 RCEPS
2. 64 MFCC
3. 32 Linear predictive coefficients
4. 32 Low-frequency Fourier magnitudes
5. 16 Rolloff
6. 1 Linear prediction error
7. 1 Zero-crossing rate

This concatenation has 402 dimensions of frame-level features, so that our meta-feature had $402 \times 2 = 804$ dimensions, because each feature has mean and variance. As in Section 4.3.4 all results were estimated by 5-fold cross-validation.

Figure 4.17 plots the segment and song classification rates as in Section 4.3.4. As before, we observe the divergence of segment and song rates. The song rate rises to a fold-mean of about 83% on 2.3-second segments. The fold variance is too large to say exactly which segment length was best for song classification, but it appears that any choice less than 4 seconds and greater than 2 seconds is near-optimal. The segment rate drops steadily as the segment length shrinks.

It is also interesting to have a look at the actual times for training and testing. AB.Single and AB.Multi share the same time and space complexity, and even

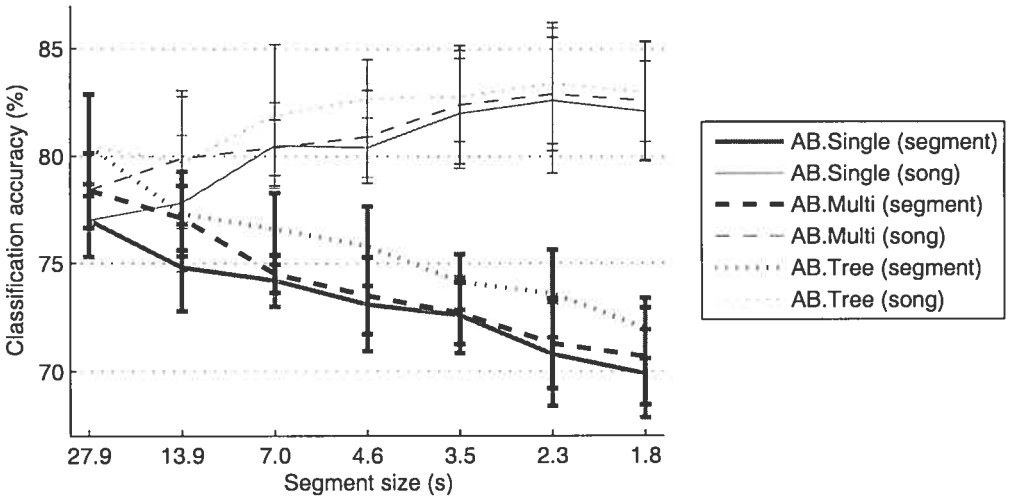


Figure 4.17: The performance of AdaBoost with concatenated feature sets, varying as a function of the segment length.

in implementation their times are similar. AB.Tree is slightly better than the two, but has a much longer training time.

Table 4.8 summarizes the accuracy, training and testing times for segments of 2.3 seconds. Note that although training for 10^4 iterations is quite slow even for our dataset, the classifications can still be done quickly. Figure 4.18 shows the song test-error curves of the two weak learners on this dataset, on a logarithmic scale. The rightmost portion of the curve is quite flat, so that if classification speed were a concern we could speed up classification ten-fold at only a small classification-rate penalty.

4.4.2 Genre Classification

Two datasets were used to evaluate the submissions: Magnatune[3], and USPOP[4]. The first database has a hierarchical genre taxonomy, with 10 classes at

[3] www.magnatune.com

[4] www.ee.columbia.edu/~dpwe/research/musicsim/uspop2002.html

Weak Learner	Accuracy	Training Time (800 files)	Test Time (200 files)
AB.SINGLE	$82.6 \pm 3.4\%$	18.4 hours	16 sec
AB.MULTI	$82.9 \pm 2.6\%$	19.9 hours	16 sec
AB.TREE	$83.4 \pm 2.8\%$	216.5 hours	24 sec

Table 4.8: The results on the 2.3 second segments size for the Tzanetakis database, with training and test time.

the most detailed level. It has 1005 training files and 510 testing files. USPOP contains 940 files for training and 474 files for testing, divided into 6 genres.

Since these two are databases of whole songs (not 30 second clips as in our own experiments), each dataset provided roughly 10 times more training data than we used in our own experiments. At the same time, competition rules demand that every algorithm must train a model and make its predictions within 24 hours. In these conditions, we feared that if we used our estimated optimal segment length of 2.3 seconds, that we would generate so many training examples that ADABOOST would not have time to converge to a good model. Therefore, we chose the sub-optimal segment length of 13.9 seconds. We submitted two versions of our algorithm for the genre competition: AB.SINGLE and AB.TREE as AB.MULTI was not available at the time. The two variants ranked first and second respectively among all entries in case of the Magnatune dataset and obtained an accuracy of 77.75% (compared to an accuracy of 71.96% that was obtained by the third ranked entry [ME05a]). In case of the USPOP dataset, the tree-based learner edged out the single-threshold learner; the accuracies are, respectively, 86.92% and 86.29% (compared to an accuracy of 85.65% that was obtained by the third ranked entry [ME05a]).

Table 4.9 provides a summary of contest results averaged across the two databases. The last two entries were not scored because they did not run or timed out. More details regarding the evaluation procedure can be found at `http://www.music-ir.org/mirexwiki/index.php/Audio_Genre_Classification`. The full table of results can be found at `http://www.music-ir.org/evaluation/mirex-results/`

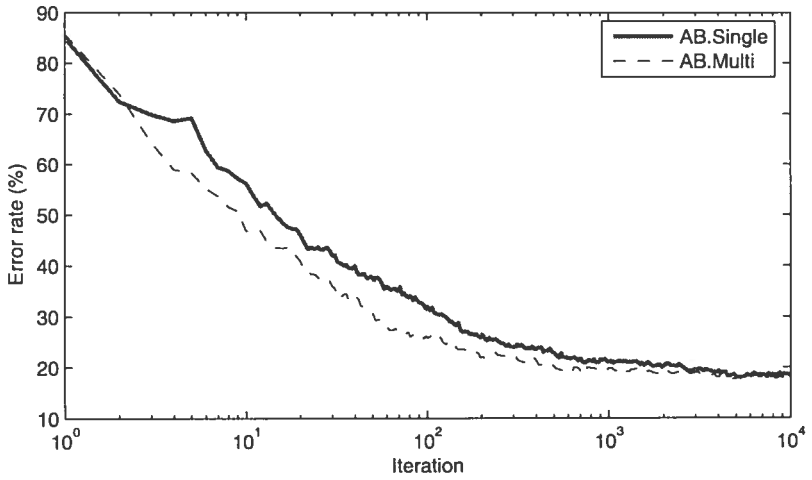


Figure 4.18: The song test-error curves of the two weak learners on 2.3 seconds segments. The multi-threshold approach converges at a faster rate, and the final results are comparable.

`audio-genre/index.html` (which includes pointers to the extended abstracts of the participating entries).

4.4.3 Artist Identification

The music data from the Genre Classification Contest were re-used for this contest, along with the experimental set-up. Each of the datasets has a total of 77 artists. However, the number of training files (1158 in each case) and testing files (642 and 653, respectively) was different.

We entered the same algorithms as above, without any modifications. The single threshold weak learner ranked first with 77.26% on the Magnatune dataset, whereas the three level tree weak learner ranked third with 74.45%. The second entry's [ME05a] performance was 76.60%. On the USPOP dataset, [ME05a] obtained 68.30%, whereas our algorithms ranked second and third, with 59.88% (single-threshold) and 58.96% (three-level tree), respectively.

Table 4.10 provides a summary of contest results averaged across the two

Rank	Participant	Overall Performance
1	**AB.Tree**	82.34%
2	**AB.Single**	81.77%
3	Mandel & Ellis	78.81%
4	West, K.	75.29%
5	Lidy & Rauber [1]	75.27%
6	Pampalk, E	75.14%
7	Lidy & Rauber [2]	74.78%
8	Lidy & Rauber [3]	74.58%
9	Scaringella, N.	73.11%
10	Ahrendt, P.	71.55%
11	Soares, V.	60.98%
12	Tzanetakis, G.	60.72%
13	Burred, J.	53.45%
*	Chen & Gao	N/A
*	Li,M.	N/A

Table 4.9: Summarized results for the Genre Recognition contest at MIREX 2005.

databases. The last three entries were not scored because they did not complete or timed out. More details regarding the evaluation procedure can be found at http://www.music-ir.org/mirexwiki/index.php/Audio_Artist_Identification. The full table of results can be found at http://www.music-ir.org/evaluation/mirex-results/audio-artist/index.html (which includes pointers to the extended abstracts of the participating entries).

4.4.4 Discussion

In the genre classification task, our algorithms obtained the highest classification rates on both databases. Since the tree weak-learner outperformed the stump weak-learner, we conjecture that the genres are hard to separate linearly in the space of the outputs of the stumps. It is interesting that the algorithm of West and Cox [WC05] is outperformed by our model, because they used a similar segmentation strategy. Their algorithm uses an onset detector to segment the music according to the onset of notes. Our algorithms perform much differently, and it could be

Table 4.10: Summarized results for the Artist Identification contest at MIREX 2005.

Rank	Participant	Overall Performance
1	Mandel & Ellis	72.45%
2	**AB.Single**	68.57%
3	**AB.Tree**	66.71%
4	Pampalk, E.	61.28%
5	West & Lamere	47.24%
6	Tzanetakis, G.	42.05%
7	Logan, B.	25.95%
*	Lily & Rauber[1]	N/A
*	Lily & Rauber[2]	N/A
*	Lily & Rauber[3]	N/A

for a number of reasons. First, context-sensitive segmentation could actually be harmful to classification, but this would be surprising. Second, the large number of features we used provides important additional information; this is supported by our results in Section 4.3.4. Third, AdaBoost on stumps and trees could be a more robust classifier for the features we are using; this is supported by the observation that in very high-dimensional spaces like our meta-feature space, Gaussians can be indiscriminate.

In the artist-recognition task, our algorithm performed well, but proved not to be as effective as the one developed by Mandel and Ellis [ME05b]. Their method was to fit a single Gaussians to a bag of MFCC frames over a whole song, and to classify using an SVM, with a similarity based on the symmetric KL divergence between these Gaussians. Their method is simple and elegant, and we wonder if it would not be more performant if applied to shorter segments of the song as in our approach. We also wonder how our own algorithm would have performed if we had used the optimal settings found in this work. We entered out algorithm with the [suboptimal] segment length of 13.9s in order to guarantee that AdaBoost would complete a sufficient number of iterations over the training data within the time limit.

Finally we were not able to enter the contest with the temporal features dis-

cussed in Section 3.3. These features are likely to increase the accuracy because they are totally uncorrelated to the timbral-based features. After the contest we ran a series of experiments with the same dataset and the tempo information obtained with the phase preserving autocorrelation. The preliminary results are summarized in Table 4.11.

Weak Learner	Accuracy (without tempo)	Accuracy (with tempo)
AB.SINGLE	$82.6 \pm 3.4\%$	$82.1 \pm 3.1\%$
AB.MULTI	$82.9 \pm 2.6\%$	$82.5 \pm 2.5\%$

Table 4.11: The results without and with tempo on the 2.3 second segments size for the Tzanetakis database. AB.TREE could not be computed for lack of time.

Surprisingly, tempo did not improve the accuracy. A possible explanation is that the tempo value extracted with the phase autocorrelation can have high probability of being either the true tempo or a multiple of it. This might confuse the classifier, which cannot find the correct one in enough examples. It is also possible that a single value cannot capture the richness of rhythmical structure which exists even within a single genre.

4.5 Conclusion

In this chapter we showed the results of our experiments on the features seen in Chapter 3 and the algorithms of Chapter 2. In the first section (4.1) we used the MNIST database to evaluate the hyper-parameters of the multi-class version of ADABOOST and its extensions.

In the following section (4.2) we showed how to use simple Haar-like features, discussed in Section 3.4 to build a robust frame-level speech/music discriminator, using an approach inspired by an algorithm used in vision.

In Section 4.3 we used standard features, discussed in Section 3.1 and Section 3.3 to build a music genre classifier, and in the following section (4.4) we described our

performance at the recent MIREX 2005 international contest in music information extraction, where our approach ranked first in audio genre classification among 15 submissions, and the second-best among 10 submission at recognizing artists.

CHAPTER 5

CONCLUSION

In this thesis, we presented several effective strategies for extracting features from audio signals, and the learning algorithms that can be used to face the challenges of automatic music classification. These strategies involve the use of a combination of meaningful features, that are extracted from the audio signal with two basic approaches:

- with the knowledge from signal processing theory, physics of sound, psychoacoustics, human auditory perception and music theory,
- with the use of a supervised learning algorithm that identifies and learns the pattern of a particular aspect of the sound, for instance the pattern of the voice or of an instrument.

The first method represents the pre-processed features discussed in Section 3.1 and Section 3.3. We built a large observation vector which included them and we used it to classify genre in acoustic data (that is, music) using a multi-class version of AdaBoost. This algorithm proved to be very effective in selecting and evaluating only the portions of the observation vector which are really meaningful for the final classification, where other algorithms fail because of the high dimensionality of the input. We discussed the origins and the development of AdaBoost, in Chapter 2, along with some of its variations. In particular we covered the multi-class extension AdaBoost.MH in Section 2.5 (and the variant AdaBoost.MO), and three variants of weak learners used by it. We investigated the different set-ups of the algorithm and the hyper-parameters, using a standard dataset in Section 4.1, and compared it with other algorithms.

Another aspect that proved to be a key factor for a successful music classifier, is the segmentation of the music into fragments that are evaluated by the classifier, seen in Section 3.2 and Section 4.3. Once trained, AdaBoost.MH returns class

posteriors for each segment, that is its classwise vote. The sum of the contributions of each segment does result in a vector from which the winner is chosen. This strategy alone increases the accuracy by 5% and reduces the size of the observation vector.

Our approach demonstrated to be effective on three databases in musical genre prediction, showing rates of 77% on Magnatune database, 83% on Tzanetakis' database, and of 87% on USPOP database. To our knowledge, at the time of writing these results are the highest genre prediction rates published for all three databases.

AdaBoost is also the core algorithm used for the second approach in feature selection. We used it to build a classifier capable of distinguishing geometrical patterns inside a visual rendering of sound. This approach allows to create fast and flexible "features" modeled on a binary input, like *music–versus–speech* seen in Section 4.2, but also *trumpet–versus–rest of the orchestra* or, more generally, *event–versus–background sound.* The resulting algorithm yielded the best performance to date on single frame classification for speech/music discrimination, when directly compared to other frame level features.

5.1 Future Work

Specifically related to the aspects of this work, there are some open questions that we would like to address in the future. Here is a partial list of these tasks:

- Develop and test more rhythmical-related features. We have been using a single value that belongs to this group of features, that is the tempo. But music involves temporal structures of much higher complexity that are extremely important for defining musical style similarity.

- Improve the segmentation technique discussed in Section 3.2 and Section 4.3.2. It would be interesting to evaluate our algorithm and features with segments created using West and Cox's [WC05] segmentation algorithm, partition the song into a set of segments that correspond to instruments playing notes.

- Add other features. AdaBoost has proved to be very effective in evaluating and selecting only the features needed for the classification. Adding other features will likely increase the accuracy, where other algorithms would suffer from the curse of dimensionality if the number of features is too large.

- Prune the features. Again, using AdaBoost it is possible to identify the elements of the features (or the features themselves) that are not effectively contributing to the classification. These elements are the ones which have not been chosen by AdaBoost during training or which have a very low confidence parameter α. Once they have been identified, these dimensions can just be ignored, speeding up considerably the training time. Future research will focus on the viability of this approach, which must be robust to the changes of dataset.

- Create of a database of segments of music, in which each instrument is labeled. With the Haar-like features approach it should be possible to build instrument-specific features, that would highlight the sections of the signal in which the instrument is present.

- Reduce the training time for Haar-like features. A large amount of time is spent recomputing similar features and sorting their vector projections. If the system has enough memory, it is possible to pre-compute the feature's outputs and sort them, before the boosting algorithm starts.

- Another approach to solve the previous problem is to map each feature's output to a bin, effectively downsampling the vector projections. This would make it possible to store all the outputs on a much smaller space, although it is not clear how the downsampling would impact the classification.

Using machine learning to solve problems related to music is a relatively recent field of research. In our work we have investigated a very small aspect of it, and even the problems that we addressed warrant further research.

In a Turing-test conducted by Soltau in 1997 [Sol97], 37 subjects trained to distinguish rock from pop performed at a level of similar performances with his automatic classifier. However, Lippens et al. [LMDM04] showed that the expected performance among human beings, for the genre classification problem, is around 90%.

This suggests that the genre recognition algorithms have not yet reached the baseline, but are potentially good to extend to more difficult, but more practical music classification tasks, such as playlist generation and music recommendation.

BIBLIOGRAPHY

[ABR64] M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821 – 837, 1964.

[AMB02] Jitendra Ajmera, Iain A. McCowan, , and Hervé Bourlard. Robust HMM based speech/music segmentation. In *Proceedings of ICASSP-02*, 2002.

[AP03] J.J. Aucouturier and F. Pachet. Representing musical genre: A state of the art. *Journal of New Music Research*, 32(1):1–12, 2003.

[ASS00] Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. In *Proceedings 17th International Conference on Machine Learning*, pages 9–16. Morgan Kaufmann, San Francisco, CA, 2000.

[BCE+05] J. Bergstra, N. Casagrande, D. Erhan, D. Eck, and B. Kegl. Predicting music genres using multi-timescale feature extraction and adaboost. *Machine Learning*, 2005. Submitted.

[Bel61] Richard E. Bellman. *Adaptive Control Processes.* Princeton University Press, Princeton, NJ, 1961.

[Ber05] James Bergstra. MSL: The Montreal Scientific Library, 2005. http://savannah.nongnu.org/cvs/?group=libmsl.

[Bis95] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[Bre96] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[CB01] Ronan Collobert and Samy Bengio. SVMTorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001.

[CEK05a] N. Casagrande, D. Eck, and B. Kegl. Geometry in sound: A speech/music audio classifier inspired by an image classifier. In *Proceedings of the International Computer Music Conference (ICMC)*, 2005.

[CEK05b] Norman Casagrande, Douglas Eck, and Balázs Kégl. Frame-level audio feature extraction using AdaBoost. In *Proc. 6th International Conference on Music Information Retrieval (ISMIR 2005)*, pages 345–350, 2005.

[CS05] T. Crawford and M. Sandler, editors. *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR 2005)*, 2005.

[DB95] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[DHS01] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* John Wiley & Sons, Inc., second edition, 2001.

[Die00] Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000.

[EC05] Douglas Eck and Norman Casagrande. Finding meter in music using an autocorrelation phase matrix and Shannon entropy. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR 2005)*, pages 504–509, 2005.

[EeGR00] Gerard Escudero and Lluis Màrquez et German Rigau. Boosting applied to word sense disambiguation. In *Proceedings of the 12th European Conference on Machine Learning, ECML. Barcelona*, 2000.

[ER02] H. Ezzaidi and J. Rouat. Speech, music and songs discrimination in the context of handsets variability. In *Proceedings of ICSLP 2002*, pages 16–20, 2002.

[Fre95] Yoav Freund. Boosting a weak learning algorithm by majority. In *Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*, 1995.

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[HMBK92] Hynek Hermansky, Nelson Morgan, Arun Bayya, and Phil Kohn. RASTA-PLP speech analysis technique. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1992.

[HTF01] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, August 2001.

[LBBH98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[LBPR01] Einat Liebenthal, Jeffrey R. Binder, Rebecca L. Piorkowski, and Robert E. Remez. Sinewave speech/nonspeech perception: An fmri study. *The Journal of the Acoustical Society of America*, 109(5):2312–2313, May 1, 2001.

[LLZ03] Dan Liu, Lie Lu, and Hong-Jiang Zhang. Automatic mood detection from acoustic music data. In *Proceedings of the International Conference on Music Information Retrieval*, 2003.

[LMDM04] S. Lippens, J.P. Martens, and T. De Mulder. A comparison of human and automatic musical genre classification. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages iv–233– iv–236, 2004.

[LT03a] Tao Li and George Tzanetakis. Factors in automatic musical genre classification. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2003.

[LT03b] Tao Li and George Tzanetakis. Factors in automatic musical genre classification of audio signals. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2003.

[LTMM04] Stefaan Lippens, George Tzanetakis, Jean-Pierre Martens, and Tom De Mulder. A comparison of human and automatic musical genre classification. In *Proceedings 2004 IEEE International Conference on Acoustic Speech and Signal Processing (ICASSP)*, 2004.

[Mak75] J Makhoul. Linear prediction: A tutorial review. In *Proceedings of the IEEE*, volume 63, pages 561–580, 1975.

[MAL05] A. Meng, P. Ahrendt, and J. Larsen. Improving music genre classification by short-time feature integration. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, mar 2005.

[MBBF00] L. Mason, P. Bartlett, J. Baxter, and M. Frean. Boosting algorithms as gradient descent. In *Advances in Neural Information Processing Systems*, volume 12, pages 512–518. The MIT Press, 2000.

[ME05a] Michael Mandel and Dan Ellis. Song-level features and SVMs for music classification. Extended Abstract, 2005. MIREX 2005.

[ME05b] Michale I. Mandel and Daniel P.W. Ellis. Song-level features and support vector machines for music classification. In *ISMIR 2005 6th International Conference on Music Information Retrieval*, 2005.

[MF05] C. McKay and I. Fujinaga. Automatic music classification and the importance of instrument identification. In *Proceedings of the Conference on Interdisciplinary Musicology (CIM05), Montreal, Canada*, 2005.

[MV00] Francesco Masulli and Giorgio Valentini. Effectiveness of error correcting output codes in multiclass learning problems. *Lecture Notes in Computer Science*, 1857:107+, 2000.

[Par94] R. Parncutt. A perceptual model of pulse salience and metrical accent in musical rhythms. *Music Perception*, 11:409–464, 1994.

[PE85] D.J Povel and Peter Essens. Perception of temporal patterns. *Music Perception*, 2:411–440, 1985.

[PG99] D. Perrot and Robert Gjerdigen. Scanning the dial: An exploration of factors in identification of musical style. In *Society for Music Perception and Cognition*, page 88, 1999.

[POP98] Constantine P. Papageorgiou, Michael Oren, and Tomaso Poggio. A general framework for object detection. *International Conference on Computer Vision*, 1998.

[RA04] Sourabh Ravindran and David V. Anderson. Boosting as a dimensionality reduction tool for audio classification. In *Proceedings of the 2004 International Symposium on Circuits and Systems*, volume 3, pages 465–468, 2004.

[Sau96] John Saunders. Real-time discrimination of broadcast speech/music. In *Proceedings IEEE Int. Conf. on Acoustics, Speech, Signal Processing (Atlanta, GA)*, 1996.

[Sch90] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[Sch98] E. D. Scheirer. Tempo and beat analysis of acoustic musical signals. *Journal Acoustic Society of America*, 103(1):588–601, 1998.

[SFBL97] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Proc. 14th International Conference on Machine Learning*, pages 322–330. Morgan Kaufmann, 1997.

[Sol97] Hagen Soltau. Erkennung von Musikstilen. Master's thesis, Universitat Karlsruhe, 1997.

[SS97] Eric Scheirer and Malcolm Slaney. Construction and evaluation of a robust multifeature speech/music discriminator. In *Proceedings of the 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1997.

[SS98] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *COLT' 98: Proceedings of the eleventh annual conference on Computational learning theory*, pages 80–91, New York, NY, USA, 1998. ACM Press.

[TC02] George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, Jul 2002.

[TdSL00] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

[TEC01] G. Tzanetakis, G. Essl, and P. Cook. Automatic musical genre classification of audio signals. In *Proceedings of the Second International Conference on Music Information Retrieval (ISMIR 2001)*, 2001.

[TEC02] George Tzanetakis, Andrey Ermolinskyi, and Perry Cook. Pitch histograms in audio and symbolic music information retrieval. In Michael

Fingerhut, editor, *Proceedings of the Third International Conference on Music Information Retrieval: ISMIR 2002*, pages 31–38, Oct 2002.

[TV01] Kinh Tieu and Paul A. Viola. Boosting image retrieval. In *Proceedings of Computer Vision and Pattern Recognition*, volume 1, pages 228–235, 2001.

[Val84] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

[Vap99] Vladimir Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, 1999.

[VJ01] Paul A. Viola and Michael J. Jones. Robust real-time object detection. *International Conference on Computer Vision*, page 747, 2001.

[WC05] Kris West and Stephen Cox. Finding an optimal segmentation for audio genre classification. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR 2005)*, 2005.

[WE99] Gethin Williams and Daniel P.W. Ellis. Speech/music discrimination based on posterior probability features. In *Proceedings of European Conference on Speech Communication and Technology*, pages 687–690, Sept. 1999.

[WEBS05] Jason Weston, Andre Elisseeff, Gökhan Bakir, and Fabian Sinz. The spider machine learning library, 2005. http://www.kyb.tuebingen.mpg.de/bs/people/spider/.

[XCST03] Changsheng Xu, Namunu C.Maddage, Xi Shao, and Qi Tian. Musical genre classification using support vector machines. In *In International Conference of Acoustics,Speech & Signal Processing (ICASSP03)*, 2003.