Filter Trees for Combining Binary Classifiers

M.Phil. Computer Speech, Text and Internet Technology
Project report

Christopher Gautier

Churchill College
University of Cambridge

Under the supervision of Dr. Mark Gales

June 18, 2008

Abstract

The filter tree, a new reduction scheme for the N-class classification problem, has been recently developed by Beygelzimer *et al.* Filter trees have been proven to be both robust and consistent. In this report, we investigate filter trees whose nodes are binary SVM classifiers, and whether they can be applied to the problem of small-vocabulary, continuous ASR. A SVM-based filter tree was implemented for the ISOLET and AURORA tasks, and demonstrated encouraging improvements over a selection of baseline systems.

Declaration

I, Christopher Gautier, a candidate for the M.Phil. in Computer Speech, Text and Internet Technology, hereby declare that this dissertation is the result of my own work except where explicitly specified, and that it does not contain material that has already been used for any comparable purpose. The total word count, which includes the footnotes, the appendices and the bibliography, is less than 15,000 words, as prescribed in the Regulations for the M.Phil. examination.

Date:

Total word count: 14745

Acknowledgments

I wish to thank my supervisor, Mark Gales, for coming up with the project idea, and regularly nudging me towards interesting lines of investigation. I also would like to thank Chris Longworth for the software he provided me, which greatly helped me getting into the subject. Finally, the project was indirectly but significantly made easier thanks to Nathan Smith's and Thorsten Joachims's works.

Lecture notes from Mark Gales and Phil Woodland were used while writing chapters 2 and 3. Errors, if any, are my own. In the subsequent chapters, specific references are given where relevant.

Contents

Chapter 1

Introduction

Binary classifiers have been thoroughly studied in the past, especially in the context of pattern recognition. Popular techniques such as linear classifiers, support vector machines (SVMs) and neural networks (notably perceptrons) are now well understood.

However, many pattern recognition problems, such as automatic speech recognition (ASR), are essentially multi-class. Most of the binary classifiers cannot readily be extended to handle more than two classes. Therefore, devising multi-class recognisers by using binary classifiers as building blocks is currently an active subject of research. The filter tree algorithm is a relatively new approach that has been proven to be both consistent and robust, compared to other known reduction schemes.

Noisy environments constitute a thorny obstacle against reliable speech recognition. The continuous noise in a car may disturb the navigation system; loud, intermittent noises in a factory can incapacitate a voice-controlled command panel. Noise perturbs the recorded observations, and introduces a mismatch between the (usually clean) training conditions, and the environment where the ASR system is effectively put to use. Even a relatively clear signal-to-noise ratio (SNR) of 20dB can severely damage the accuracy of recognisers that do not perform some form of noise compensation. State-of-the-art ASR systems can achieve laudable results, but there is still room for improvement.

This report is two-fold. First, the feasibility of using SVM-based filter trees for speech recognition is examined. One issue is that the dynamic audio data must be converted into static data usable by SVMs. It is also unclear what kind of feature space should be used at each node. Secondly, the accuracy of the filter tree algorithm is evaluated in the context on two small-vocabulary ASR tasks: a clean isolated letter database (ISOLET), and a noisy continuous digit database (AURORA).

Chapter 2

ASR with Hidden Markov Models

The Hidden Markov Model (HMM) is a well-known generative model of speech. One reason for its popularity is that HMMs are naturally apt to classify variable-length data. As they are a fundamental tool in this report, HMMs are briefly discussed in this chapter.

2.1 HMMs as generative models of speech

Given a sequence of observation vectors $\mathbf{O} = (\vec{o}_1, \ldots, \vec{o}_T)$, the problem of automatic speech recognition is to find the most likely sequence of words $\hat{\mathbf{W}} = (w_1, \ldots, w_n)$:

$$\hat{\mathbf{W}} = \underset{W}{\operatorname{argmax}}\, P(\mathbf{W}|\mathbf{O}) = \underset{W}{\operatorname{argmax}}\, \frac{P(\mathbf{O}|\mathbf{W})P(\mathbf{W})}{P(\mathbf{O})} = \underset{W}{\operatorname{argmax}}\, P(\mathbf{O}|\mathbf{W})P(\mathbf{W})$$

$P(\mathbf{O}|\mathbf{W})$ is called the *acoustic model*, and quantifies the likelihood of an observation given a certain word. The *language model* $P(\mathbf{W})$ predicts the probability of a sequence of words in a given language.

HMMs are generative models for the acoustic model. They are a kind of finite-state machines defined by:

- a set of states $\{s_i\}$
- a transition matrix (a_{ij}) where the constant a_{ij} is the probability to go from state s_i to s_j
- a set of output probabilistic functions $\{b_i(\vec{o})\}$, one per state

Because HMMs are often used to represent temporal phenomena, the state topology is typically a left-to-right one, possibly with no skip, although this is not by all means always the case. Figure 2.1 shows a typical

representation of a HMM. HMMs also include a non-emitting start and end states. One key advantage of these two states is that it is easy to glue HMMs together into complex networks.

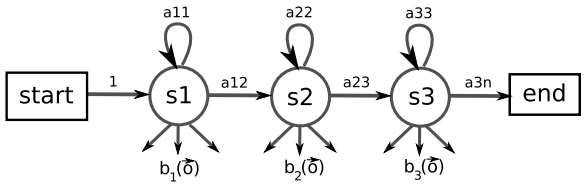

Figure 2.1: A typical HMM

Considering a time discretisation into frames of constant duration, at each frame t, the HMM changes its active state from s_i to s_j, depending on the (a_{ij}), then emits a feature vector $\vec{o}_t$ with probability $b_j(\vec{o}_t)$. The Markov chain is *hidden* in the sense that the current active state is not known; only the $(\vec{o}_t)$ can be observed.

The b_i functions are typically mixtures of M d-dimensional Gaussian components, defined by M means $\vec{\mu}_{im}$ and M covariance matrices $\mathbf{\Sigma}_{im}$:

$$b_i(\vec{o}) = \sum_{m=1}^{M} \frac{c_{im}}{(2\pi)^{d/2}|\mathbf{\Sigma}_{im}|^{1/2}} e^{-\frac{1}{2}(\vec{o}-\vec{\mu}_{im})^\top \mathbf{\Sigma}_{im}^{-1}(\vec{o}-\vec{\mu}_{im})}$$

As generative models of speech, HMMs can be applied to speech recognition, by finding the most likely sequence of HMM states through an HMM network. Efficient dynamic programming procedures exist to do that, a popular one being the token-passing Viterbi algorithm.

2.2 Expectation-Maximisation

It should be noted that Gaussian mixture models can be seen as networks of single-component HMMs. Figure 2.2 illustrates this. In this section, we therefore assume only one Gaussian component per state, in order to simplify the discussion.

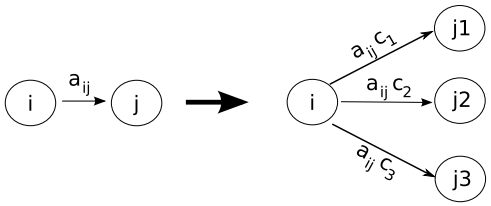

Figure 2.2: If state j has 3 Gaussian components with weights c_1, c_2, c_3, the HMM can be replaced by a network of single-components HMM.

Let θ be the set of all the HMM model parameters, *i.e.* the set $\{\vec{\mu}_i, \mathbf{\Sigma}_i, a_{ij}\}$ of all means and variances of the Gaussian component for each emitting state, as well as the transition matrix.

Training a HMM is about estimating θ. This must be done algorithmically, since the high dimensionality of θ precludes a manual intervention. Maximum Likelihood Estimation (MLE) methods are often used to train HMMs. They aim at maximising $P(\mathbf{O}|\theta)$ for a given set of training data. Direct maximisation of P is not possible with HMMs though, because the exact sequence of states that generated $\mathbf{O}$ is unknown. Dempster *et al.* introduced in [5] the Expectation-Maximisation (EM) iterative optimisation algorithm, which overcomes the issue.

Let $\mathcal{Z}$ be the set of all possible sequences of HMM states, and $\theta^{(k)}$ be the estimated value of θ at the k^{th} iteration.

First, the auxiliary function is introduced as:

$$Q(\theta^{(k)}, \theta^{(k+1)}) = \sum_{\mathbf{S} \in \mathcal{Z}} P(\mathbf{S}|\mathbf{O}, \theta^{(k)}) \log P(\mathbf{O}, \mathbf{S}|\theta^{(k+1)})$$

It can be shown that increasing the auxiliary function will increase the likelihood:

$$Q(\theta^{(k)}, \theta^{(k+1)}) \geq Q(\theta^{(k)}, \theta^{(k)}) \Rightarrow P(\mathbf{O}|\theta^{(k+1)} \geq P(\mathbf{O}|\theta^{(k)})$$

The EM algorithm consists of two steps:

1. E step: Compute the expectancy $\mathcal{E}_{\mathbf{S} \in \mathcal{Z}}\Big(\log P(\mathbf{O}, \mathbf{S}|\theta^{(k+1)}) \Big| \mathbf{O}, \theta^{(k)} \Big)$
2. M step: Maximise $Q'(\theta^{(k+1)}) = Q(\theta^{(k)}, \theta^{(k+1)})$ with respect to $\theta^{(k+1)}$

The EM process should be repeated until convergence has been reached. The mathematical foundations described in [5] guarantee that each iteration of the process will not decrease the likelihood, but it is possible that it converges only towards a local maximum.

Note that the process must be somehow initialised by estimating $\theta^{(0)}$. When training HMMs, this can be done by computing the means $\vec{\mu}$ and variance $\mathbf{\Sigma}$ of the training data, and initialise all the Gaussians' means and variances with the same values $\vec{\mu}$ and $\mathbf{\Sigma}$. The formulæ to update the components' means and variances, derived from the EM algorithm, are:

$$\begin{cases} \vec{\mu}_i^{(k+1)} = \dfrac{\sum_{t=1}^{T} \gamma_i^{(k)}(t)\vec{o}(t)}{\sum_{t=1}^{T} \gamma_i^{(k)}(t)} \\ \mathbf{\Sigma}_i^{(k+1)} = \dfrac{\sum_{t=1}^{T} \gamma_i^{(k)}(t)(\vec{o}(t) - \vec{\mu}_i^{(k+1)})(\vec{o}(t) - \vec{\mu}_i^{(k+1)})^\top}{\sum_{t=1}^{T} \gamma_i^{(k)}(t)} \\ a_{ij}^{(k+1)} = \dfrac{\sum_{t=1}^{T} \alpha_i^{(k)}(t) a_{ij}^{(k)} b_j^{(k)} \vec{o}(t+1)\beta_j^{(k)}(t+1)}{\sum_{t=1}^{T} \alpha_i^{(k)}(t)\beta_i^{(k)}(t)}, \text{for } j < N \end{cases}$$

where $\alpha_i^{(k)}$, $\beta_i^{(k)}$ and $\gamma_i^{(k)}$ are functions defined as:

$$\begin{cases} \alpha_i^{(k)}(t) = P(\vec{o}(1), \ldots, \vec{o}(t), s(t) = i | \theta^{(k)}) \\ \beta_i^{(k)}(t) = P(\vec{o}(t+1), \ldots, \vec{o}(T) | s(t) = i, \theta^{(k)}) \\ \gamma_i^{(k)}(t) = P(s(t) = i | \mathbf{O}, \theta^{(k)}) \end{cases}$$

2.3 Large vocabulary tasks

A difficult class of ASR problems relates to the support of vocabulary larger than, say, 10^4 words. Building one HMM for each word of the vocabulary is not a scalable solution. However, HMMs can be built not only for words, but also for a variety of sub-words units, such as phones, or *triphones* (phones with the immediate left and right context).

Suppose a pronunciation dictionary, or *lexicon*, is available. The problem of large vocabulary continuous speech recognition (LVCSR) can addressed by:

- Building and training HMMs for a set of 40 to 50 phones
- Recognition yields the most probable sequence of phones
- By looking up the pronunciations in the lexicon, the word sequence can be reconstructed

HMMs have been shown to be an efficient and flexible solution to the LVCSR problem.

Chapter 3

Support vector machines

3.1 Actual risk and empirical risk

3.1.1 Notion of risk

Let $\mathbb{X}$ be a finite set of l points $\{\vec{x}_1, \ldots, \vec{x}_l\}$ of $\mathbb{R}^d$, where each point is given a binary label $y \in \{0, 1\}$. Consider a set $\{f_\theta(\vec{x}) : \mathbb{X} \to \{0, 1\}\}$ of binary classifiers, parametrised by some variable θ. The *risk* $R(\theta)$ of a classifier $f_\theta(\vec{x})$ is defined as the expected classification error rate:

$$R(\theta) = \iint |y - f_\theta(\vec{x})| p(\vec{x}, y) d\vec{x} dy$$

Because $p(\vec{x}, y)$ is rarely known, we can only calculate the *empirical risk* over a finite set of l points:

$$R_{emp}(\theta) = \frac{1}{l} \sum_{i=1}^{l} |y_i - f_\theta(\vec{x}_i)|$$

One shortcoming of the empirical risk is that it is only a lower bound of the actual risk. It is indeed easy to train a classifier so that $R_{emp} = 0$ on the training data, while the actual error rate on the test data is high[1].

The sections below show that the empirical risk can still be used to estimate the actual risk.

3.1.2 VC dimension

A set S of binary classifiers $\{f_\theta\}$ is said to *shatter* a subset $\mathbb{X}$ of $\mathbb{R}^d$ if, for every binary labelling $y(x) : \mathbb{X} \to \{0, 1\}$ of $\mathbb{X}$, there exists a classifier $f^* = f_{\theta^*}$ that correctly classifies all samples of $\mathbb{X}$. The *Vapnik-Chervonenkis*

[1] One degenerate example is simply to build a hash table for all the training data, provided that it contains no mislabelled sample. The generalising power of such a classifier is of course null, because it is unable to classify any unseen example.

(VC) dimension h of S over $\mathbb{X}$ is defined as the maximum number of points drawn from $\mathbb{X}$ that can be shattered by S.

To illustrate this definition, let d be equal to 2, $\mathbb{X}$ be a set of non-collinear points of $\mathbb{R}^d$, and S be the set S_2^l of all the linear classifiers over $\mathbb{R}^2$. Figure 3.1 graphically demonstrates that, for each possible triplet T of labelled points, there exists a linear decision boundary that shatters T. A counter-example is given in figure 3.2, where a set of four labelled samples cannot be shattered by any classifier of S_2^l. That means that the VC dimension[2] of S_2^l is 3.

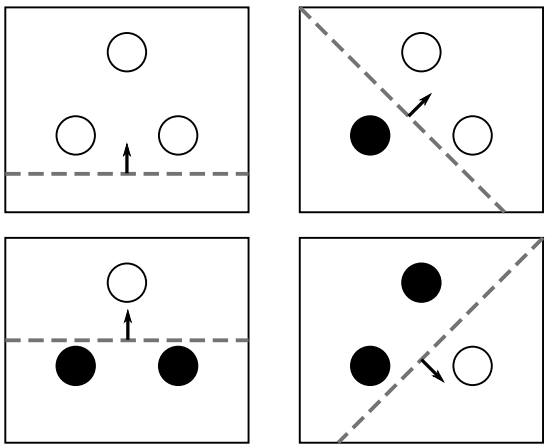

Figure 3.1: S_2^l shatters any labelled, linearly-independent triplet of $\mathbb{R}^2$

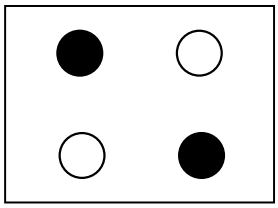

Figure 3.2: A counter-example with four points of $\mathbb{R}^2$

Informally, the VC dimension h can be interpreted as a measure of the complexity of a learning machine. Yet, it is interesting to note that decreasing the dimensionality of θ does not necessarily decrease the VC dimension. Burges [3] gives an example of $S = \{f_\theta\}$ where θ is a plain real number, but the VC dimension of S is infinite. Similarly, an infinite VC dimension is not a guarantee of good classification performances.

[2]Note: if the three points are aligned, then it may not be possible to shatter them with a linear classifier. Consider the case where the labels of the three aligned points are respectively 0, 1 and 0.

3.1.3 Structural risk minimisation

As shown by Vapnik[24], one of the practical uses of the VC dimension h is to provide an upper bound of the actual risk $R(\theta)$, as a function $b(h, l, \eta)$ so that the following inequality holds with a probability of $1-\eta$, with $\eta \in]0, 1[$:

$$R(\theta) \leq R_{emp}(\theta) + b(h, l, \eta)$$

The exact expression of $b(h, l, \eta)$ is beyond the scope of this report, but it is schematically represented in figure 3.3.

The structural risk minimisation (SRM) principle, as introduced by Vapnik, is to choose the classifier f_θ that minimises the bound on risk. It addresses the issue of overfitting by striking a balance between the model's complexity, and its ability to fit the training data.

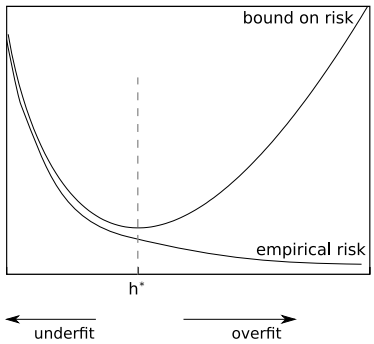

Figure 3.3: Finding the optimal VC dimension

3.2 Support vector machines

Support vector machines (SVMs) are a type of binary classifiers that approximates the SRM principle.

Consider the general problem of finding a binary classifier for a training set of labelled[3] samples $\mathbb{X} = \{(\vec{x}_i, y_i)\}$, where:

$$\forall i, \begin{cases} \vec{x}_i \in \mathbb{R}^d \\ y_i \in \{-1, 1\} \end{cases}$$

3.2.1 The linearly-separable case

First, we assume that the points are linearly separable, which means that there exists at least one hyperplane $\mathcal{H}$ of $\mathbb{R}^d$, whose equation is of the form

[3] We took the liberty of changing the label space in order to make the later equations easier.

(3.1), which splits the space into one subspace containing all the positive examples $\mathbb{X}_+ = \{\vec{x}_i \,/\, y_i = 1\}$, and one subspace containing all the negative examples $\mathbb{X}_- = \{\vec{x}_i \,/\, y_i = -1\}$, without errors.

$$\mathcal{H} = \{\vec{x} \in \mathbb{R}^d \,/\, \vec{\phi} \cdot \vec{x} + b = 0\} \tag{3.1}$$

$\vec{\phi}$ is the oriented normal of the hyperplane $\mathcal{H}$, and b is its bias. Assuming that $\mathbb{X}_+$ is located in the half-space $\{\vec{x} \,/\, \vec{\phi} \cdot \vec{x} + b > 0\}$, then let $\mathcal{H}_+$ and $\mathcal{H}_-$ be defined as:

$$\begin{cases} \mathcal{H}_+ = \{\vec{x} \in \mathbb{R}^d \,/\, \vec{\phi} \cdot \vec{x} + b_+ = 0\} \\ \mathcal{H}_- = \{\vec{x} \in \mathbb{R}^d \,/\, \vec{\phi} \cdot \vec{x} + b_- = 0\} \end{cases}$$

where:

$$\begin{cases} b_+ = \max\limits_{x \in \mathbb{X}_+} (-\vec{\phi} \cdot \vec{x}) \\ b_- = \min\limits_{x \in \mathbb{X}_-} (-\vec{\phi} \cdot \vec{x}) \end{cases}$$

$\mathcal{H}_+$ and $\mathcal{H}_-$ classify $\mathbb{X}$ according to:

$$\forall x_i \in \mathbb{X}, \begin{cases} \vec{\phi} \cdot \vec{x} + b_+ \geq 0 & \Rightarrow x_i \in \mathbb{X}_+ \\ \vec{\phi} \cdot \vec{x} + b_- \leq 0 & \Rightarrow x_i \in \mathbb{X}_- \end{cases}$$

Figure 3.4 shows two possible choices for the decision boundary in $\mathbb{R}^2$. The rationale behind SVMs is to maximise the margin between the two hyperplanes $\mathcal{H}_+$ and $\mathcal{H}_-$. Intuitively, by achieving maximum separation between the two classes, the resulting classifier will generalise better to unseen data. For that reason, SVMs are sometimes referred to as *maximum margin* classifiers.

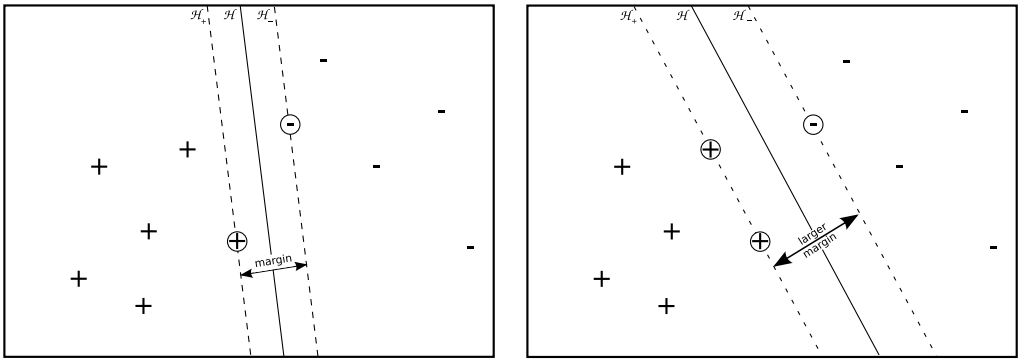

Figure 3.4: Different hyperplanes give different margins. Vectors that lie on the boundaries $\mathcal{H}_+$ and $\mathcal{H}_-$ are circled.

If $\mathcal{H}$ is equidistant from $\mathcal{H}_+$ and $\mathcal{H}_-$, and if the input space is translated so that $\mathcal{H}$ goes though the origin, then the hyperplane equations can be rewritten as:

$$\begin{cases} \mathcal{H}_+ = \{\vec{x} \in \mathbb{R}^n \,/\, \vec{\phi} \cdot \vec{x} + b = 1\} \\ \mathcal{H}_- = \{\vec{x} \in \mathbb{R}^n \,/\, \vec{\phi} \cdot \vec{x} + b = -1\} \end{cases}$$

The distance from $\mathcal{H}_+$ and $\mathcal{H}_-$ to the origin are respectively $|1-b|/||\vec{\phi}||$ and $|1+b|/||\vec{\phi}||$. The margin between the two hyperplanes is therefore equal to $2/||\vec{\phi}||$. The problem of finding the optimal hyperplane can thus be formalised as follows:

Find $(\vec{\phi}^*, b^*)$ that minimises $\vec{\phi} \cdot \vec{\phi}$ under the following constraints:

$$\forall x_i \in \mathbb{X}, \begin{cases} \vec{\phi} \cdot \vec{x}_i + b & \geq & 1, & \text{if } y_i = 1 \\ \vec{\phi} \cdot \vec{x}_i + b & \leq & -1, & \text{otherwise} \end{cases}$$

To solve this problem, the Lagrange function is introduced:

$$L(\vec{\phi}, b, \alpha_1, \ldots, \alpha_l) = \frac{1}{2}\vec{\phi} \cdot \vec{\phi} - \sum_{i=1}^{l} \alpha_i (y_i(\vec{\phi} \cdot \vec{x}_i + b) - 1)$$

where the $\alpha_i \geq 0$ are the Lagrangian multipliers. By writing:

$$\begin{cases} \dfrac{\partial L}{\partial \vec{\phi}}(\vec{\phi}^*, b^*, \alpha_i^*) & = \vec{0} \\ \dfrac{\partial L}{\partial b}(\vec{\phi}^*, b^*, \alpha_i^*) & = 0 \end{cases}$$

one can derive:

$$\vec{\phi}^* = \sum_{i=1}^{l} \alpha_i^* y_i \vec{x}_i \tag{3.2}$$

Additionally, the Kuhn-Tucker theorem states that:

$$\forall i \in \{1, \ldots, l\},\ \alpha_i^* (y_i(\vec{\phi}^* \cdot \vec{x}_i) + b^*) - 1) = 0 \tag{3.3}$$

Equation 3.3 is interesting because it implies that the α_i^* are non zero only for samples that lie exactly on $\mathcal{H}_+$ or $\mathcal{H}_-$. Such samples are called *support vectors*. They lead to a sparse representation of the hyperplane equation for $\mathcal{H}$, because reporting the zero α_i^*'s back into equation 3.2 shows that non-support vectors do not contribute to the summation.

The optimisation problem is commonly solved for the equivalent dual optimisation problem, where the following quadratic expression is maximised:

$$W(\vec{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j \tag{3.4}$$

Because the solution can be proved to be unique, some specialised algorithms, such as [16], have been devised to efficiently solve it.

Let SV be the set of the support vectors. Once a SVM has been trained, classification is done by computing the SVM score:

$$s(\vec{x}) = \sum_{\vec{x}_i \in SV} \alpha_i^* y_i \vec{x}_i \cdot \vec{x} \tag{3.5}$$

The sign of the score gives the class of $\vec{x}$:

$$\begin{cases} s(\vec{x}) > 0 & \Rightarrow x \in \mathbb{X}_+ \\ s(\vec{x}) < 0 & \Rightarrow x \in \mathbb{X}_- \end{cases} \tag{3.6}$$

3.2.2 The non-separable case

Support vector machines can be elegantly extended to the case where the training data is not linearly separable, by using so-called *slack variables* ξ_i.

The idea is that ξ_i represents the ability for x_i to be on the wrong side of the decision boundary. As shown in figure 3.5, the ξ_i for an ill-placed sample x_i is defined as the distance from x_i to the correct hyperplan. If x_i lies in the correct half space, then ξ_i is zero.

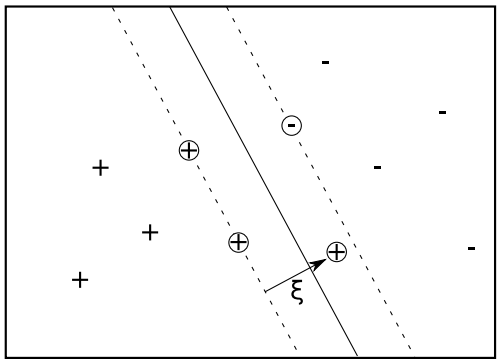

Figure 3.5: Illustration of a slack variable ξ. If ξ is in $[0, 1[$, the sample lies within the margin, but is still correctly classified. If $\xi \geq 1$, as is the case here, then the sample is misclassified by the SVM.

The optimisation problem is now about both maximising the margin *and* minimising the number of training errors. The balance between the

two conflictual objectives can be controlled by a variable C. The relaxed *soft-margin* optimisation problem is therefore defined as follows:

Find $(\vec{\phi}^*, b^*, \vec{\xi}^*)$ that minimises

$$\frac{1}{2}\vec{\phi}\cdot\vec{\phi} + C\sum_{i=1}^{l}\xi_i$$

under the following constraints:

$$\forall x_i \in \mathbb{X}, \begin{cases} \vec{x}_i\cdot\vec{\phi} + b & \geq & 1, & \text{if } y_i = 1 \\ \vec{x}_i\cdot\vec{\phi} + b & \leq & -1, & \text{otherwise} \end{cases}$$

Small values of C will maximise the margin, at the expense of more training errors.

This problem can be solved by maximising the same quadratic form 3.4 as with the hard-margin case:

$$W(\vec{\alpha}) = \sum_{i=1}^{l}\alpha_i - 1/2\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j \vec{x}_i\cdot\vec{x}_j$$

but with slightly different constraints:

$$\begin{cases} \forall i \in \{1,\ldots,l\}, & 0 \leq \alpha_i \leq C \\ \displaystyle\sum_{i=1}^{l}\alpha_i y_i = 0 \end{cases}$$

Note that, interestingly, the ξ_i do not occur in the equation. Again, optimised algorithms exist to solve this efficiently, and the run-time classification is done using equations 3.5 and 3.6, as before.

An interactive Java applet[31], based on [27], provides an entertaining visual introduction to the dynamics of support vector machines.

3.2.3 Kernels

Notice how the equations for the SVM training (3.4) and the classification score (3.5) only involve dot products of vectors. SVM can in fact be *kernelised* by substituting the bilinear dot product function by a (possibly non linear) *kernel* function.

$$\vec{x}_i\cdot\vec{x}_j \longrightarrow k(\vec{x}_i,\vec{x}_j) : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$$

Popular kernels include

- The linear kernel: $k(\vec{x}_i, \vec{x}_j) = \vec{x}_i \cdot \vec{x}_j$
- Polynomial kernels: $k(\vec{x}_i, \vec{x}_j) = (a.\vec{x}_i \cdot \vec{x}_j + b)^c$
- Gaussian kernels: $k(\vec{x}_i, \vec{x}_j) = \exp(-\gamma \, ||\vec{x}_i - \vec{x}_j||^2)$

The advantage of using kernels is that the feature space does not have to be explicited, and can potentially be of a higher dimensionality than the original input space. This is interesting because this allows SVMs to be theoretically more powerful than plain linear classifiers. As shown in figure 3.6, it is possible to map the initial input space to a better feature space, *via* a non linear mapping function Φ, so that a problem initially not separable with linear classifiers, becomes linearly separable in a judiciously augmented feature space. The mapping used in the example is:

$$\Phi\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x^2 + y^2 \\ xy \end{pmatrix}$$
$$\mathbb{R}^2 \mapsto \mathbb{R}^2$$

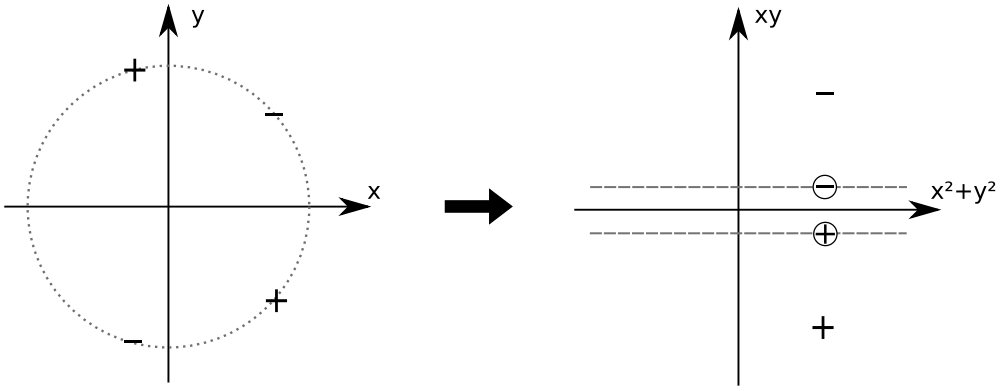

Figure 3.6: Better separation can achieved in a suitably transformed feature space.

Kernel functions are a way to introduce an implicit augmented feature space. The Mercer theorem proves the existence of a mapping $\Phi(\vec{x})$ for a given kernel k, so that $k(\vec{x}_i, \vec{x}_j) = \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$, provided that k is symmetric, continuous, and positive semi-definite. The mapping function Φ is not explicited, but it would not be useful if it was anyway.

3.3 Classifying dynamic data

SVMs are by their nature adapted to classify *static data*, that is, data of a fixed dimensionality d. Unfortunately, speech recognition involves variable-length sequences of observations. The first attempts to use SVMs in ASR systems tried to select a fixed number of observations, either by somehow picking up representative observation vectors, or by applying a dynamic time warping (DTW) transformation. Performance were not satisfactory as important information tended to be lost in the process.

3.3.1 Generative kernels

Generative kernels ([11], [21]) are derived from generative models. In the case of speech recognition, the generative models are standard HMMs. For a given variable-length sequence $\mathbf{O}$ of observation, a log likelihood score $L = \log(P(\mathbf{O}|\mathcal{M}))$ can be calculated for a given HMM $\mathcal{M}$. L is therefore a function of the means $\vec{\mu}_i$, variances $\mathbf{\Sigma}_i$, and component weights c_{im} of the HMM.

For instance, one can extract additional information from L, by deriving it with respect to some or all of the $\vec{\mu}_i$, $\mathbf{\Sigma}_i$ and c_{im} parameters. One can thus build a fixed-length feature space vector $\vec{x}$ from the variable-length sequence $\mathbf{O}$, by grouping L and the n scalars derived from L into a single vector called *score*. The set of possible scores for a given set of extracted features is called the *score space*. Scores based on the first order derivatives of L are called *Fisher scores*. Other types of scores exist. For a two-class problem, better results can be obtained[20] with score spaces defined on the log ratio:

$$LR = \log \frac{P(\mathbf{O}|\mathcal{M}_1)}{P(\mathbf{O}|\mathcal{M}_2)}$$

Generative score spaces map variable-length sequences into scores of fixed dimensionality, thus making SVMs a viable tool for ASR systems. One more refinement is needed for a efficient usage though. Due to the widely different dynamic ranges of the input data, and the fact that SVMs are sensitive to scaling, comparing vectors is not straightforward. Experiments have shown that using a plain Euclidian distance typically performs poorly.

In other words, we need to define a reasonable distance metric over the feature space. This hurdle can be overcome by applying a Mahalanobis distance based on the covariance matrix $\mathbf{\Sigma}$ of the score space. Using the improved scalar product $\Phi(\vec{a})\mathbf{\Sigma}^{-1}\Phi(\vec{b})$ *normalises* the score space by whitening it. The obtained distance is said to be *maximally non-committal* in the sense that it is not biased towards feature dimensions with large dynamic ranges.

3.3.2 Score space feature selection

Because complex HMMs can have more than 10^3 parameters, the feature vectors can quickly grow quite large. Additionally, not all the features extracted from L or LR necessarily contain useful information. It is thus worthwhile to reduce the dimensionality of the score space by dropping some of its dimensions.

A popular criterion for quantifying the usefulness of features is the Fisher ratio. For any dimension of the d-dimensional feature space, and two classes characterised by their means (μ_+, μ_-), and their variances (σ_+^2, σ_-^2), the Fisher ratio f can be expressed as the ratio of the between-class means to the within-class variances:

$$f = \frac{(\mu_+ - \mu_-)^2}{\sigma_+^2 + \sigma_-^2}$$

Intuitively, features with the largest distance between the class means, and the smallest class variances are more discriminatory. By computing the Fisher ratio for all the dimensions, and sorting them in decreasing order, a fixed number of $d' < d$ dimensions can be selected amongst the dimensions with the largest Fisher ratios.

It has been shown[21] that selecting the highest Fisher ratios does not reduce the accuracy of the SVM, and can occasionally improve the performance, by removing potentially noisy dimensions. There seems to exist an optimum number of dimensions, but it can only be determined through experimentations.

3.3.3 Score concatenation

The generative kernels discussed so far have only used one or two HMMs to build the score space. However, a typical ASR system is multi-class. In order to obtain a more expressive score space, the scores derived from each HMM can be concatenated together into a super score. Feature selection should be performed afterwards on the resulting score space, in order to prune non-discriminatory features, and keep its dimensionality reasonably low. Figure 3.7 illustrates this process with 3 HMMs.

Score space concatenation combined with score space feature selection provide a flexible framework to control the complexity of the score space while retaining the most useful information for the classification task.

3.3.4 Score space notation conventions

In this section, we describe a compact way of specifying a score space. Suppose we have two generative models $\mathcal{M}_1$ and $\mathcal{M}_2$ at our disposal. A generative score space is uniquely determined by an unordered bag of letters taken from the following set: { `l`, `s`, `u`, `m`, `v`, `w`, `x`, `y`, `z` }. Table 3.1 lists the

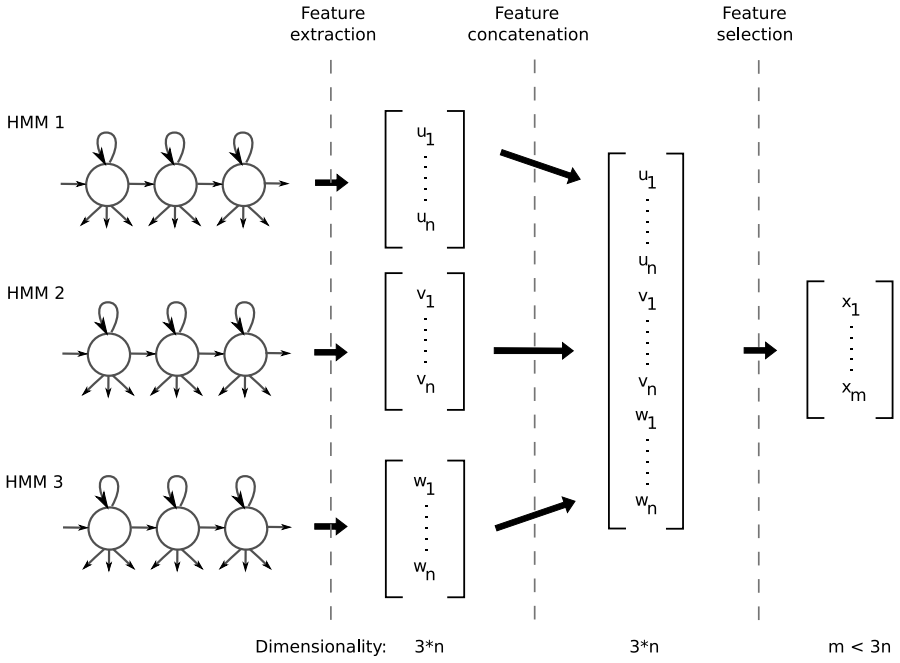

Figure 3.7: Score space concatenation and reduction

meaning of each letter. Fisher scores can be computed by considering $\mathcal{M}_2$ "constant", and focusing on the features `l`, `s`, `m`, `v` and `w` only.

The score vector is the concatenation of all the selected features. To give an example of the dimensionality, the third column of table 3.1 gives the number of dimensions for each score type, for a HMM with ten emitting states, and four 39-dimensional Gaussian components per state[4]. For example, the `lm` score space represents a 1561-dimensional vector space.

Letter	Meaning	Dimensionality
`l`	Log-likelihood ratio LR of the two models	1
`s`	$\mathcal{M}_1$ posterior P_1	1
`u`	$\mathcal{M}_2$ posterior P_2	1
`m`	Derivatives of $\mathcal{M}_1$ w.r.t. the means	1560
`v`	Derivatives of $\mathcal{M}_1$ w.r.t. the variances	1560
`w`	Derivatives of $\mathcal{M}_1$ w.r.t. the component weights	40
`x`	Derivatives of $\mathcal{M}_2$ w.r.t. the means	1560
`y`	Derivatives of $\mathcal{M}_2$ w.r.t. the variances	1560
`z`	Derivatives of $\mathcal{M}_2$ w.r.t. the component weights	40

Table 3.1: Notation system for the score space

[4]This is the HMM setup used in chapter 7.2.

Chapter 4

Multi-class classification

4.1 The N-class classification problem

Let $\mathbb{X}$ be a set of l samples, and $\mathbb{Y} = \{y_1, \ldots, y_N\}$ be the label space. Let $L(x) : \mathbb{X} \to \mathbb{Y}$ be a class distribution that assigns a unique label to each sample.

We can define a cost function $cost(x, y_i) : \mathbb{X} \times \mathbb{Y} \to \mathbb{R}^+$ that gives the cost of assigning label y_i to the sample x. The cost of correctly classifying x should be zero: $cost(x, L(x)) = 0$.

The *cost-sensitive classification problem* is the problem of finding a classifier function $C : \mathbb{X} \times \mathbb{Y}$ that minimises the classification loss $e(C, L)$:

$$e(C, L) = \sum_{x \in \mathbb{X}} cost(x, C(x))$$

It is easy to convert a cost-sensitive classification problem into a *cost-insensitive* one, by defining the cost function as:

$$\forall x \in \mathbb{X}, \forall i \in \{1, \ldots, N\}, \begin{cases} cost(x, y_i) = 1 & \text{if } L(x) \neq y_i \\ = 0 & \text{otherwise} \end{cases}$$

In that case, the loss represents the number of samples C misclassifies.

In the following sections, a selection of cost-insensitive classifiers are presented.

4.2 Multi-class SVMs

It is theoretically possible to extend the binary SVM into a N-class classifier. Given N classes, and l training samples, Weston & Watkins [26] suggest that N hyperplanes can be obtained at once by solving the following optimisation problem:

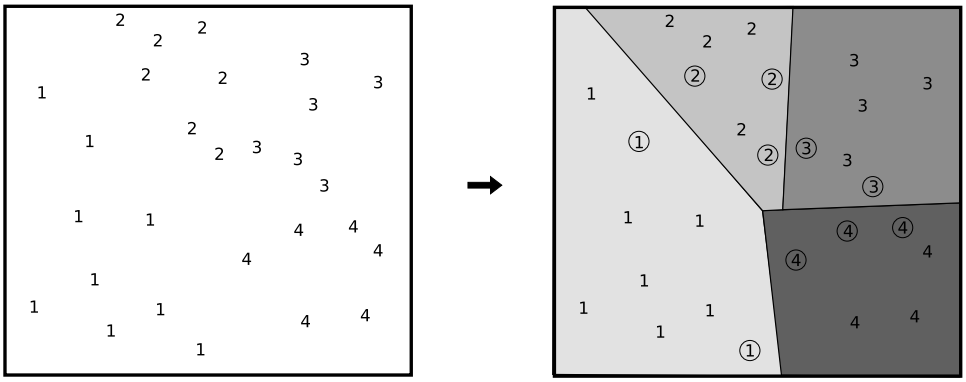

Figure 4.1: Multi-class SVM classification example. Support vectors are circled.

Find the $2N + (N-1)l$ parameters $(\vec{\phi}_n^*, b_n^*, \vec{\xi}_{n,i}^*)$ that minimise:

$$\frac{1}{2}\sum_{n=1}^{N}(\vec{\phi}_n \cdot \vec{\phi}_n) + C\sum_{i=1}^{l}\sum_{n\neq y_i}\xi_{n,i}$$

under the following constraints:

$$\forall i \in \{1,\dots,l\}, \forall n \in \{1,\dots,N\} - \{y_i\},$$
$$\begin{cases} (\vec{\phi}_{y_i} \cdot \vec{x}_i) + b_{y_i} \geq (\vec{\phi}_n \cdot \vec{x}_i) + b_n + 2 - \xi_{n,i} \\ \xi_{n,i} \geq 0 \end{cases}$$

Section 5 of [26] gives more mathematical details, including an implicit expression of the analytical solution, but no explicit solution is given.

Multi-class SVMs would appear to be a powerful tool. However, it turns out that they are not that practical to use, because the optimisation problem to solve is much more complex than its binary counterpart. Section 3 of Hsu & Lin [10] explains some of the difficulties of the task. As for today, solutions for the generic N-class SVM training are not satisfactory. In the end, it is often more profitable to train N binary SVMs, and combine their decisions into a multi-class system.

The process of splitting a multi-class classification problem into a serie of binary classifications is called a *reduction*. Many efficient binary classifiers are known today, but multi-class problems are more frequent than binary ones. This mismatch gave an incentive to study reduction schemes, a selection of which is presented in the following sections.

4.3 One-versus-one classifiers

A natural idea to reduce the N-class classification problem is to build a binary classifier for each possible pair of classes. For N classes, that means building C_N^2 binary classifiers[1].

Let (y_i, y_j) be two distinct classes, and let $[ij]$ be the associated classifier. $[ij]$ can be trained to answer the question "Is class i more probable than j for the given observation?". In that case, we say that $[ij]$ is a one-versus-one classifier.

Consider three samples x_i, x_j, x_k, representative respectively of three distinct classes y_i, y_j, y_k. An optimal $[ij]$ classifier[2] would assign the label i to x_i, and the label j to x_j. We do not care about what label $[ij]$ assigns to x_k, because the question is rather baseless. Consequently, for a given sample x_i, we have $N-1$ "meaningful" classifiers $[ij]$ (for all $j \neq i$), and $C_N^2 - N + 1$ "meaningless" classifiers.

Combining C_N^2 output labels into a final, unique label can be done by drawing an analogy with a voting process: classifying a sample is equivalent to a vote between the classifiers, to elect a winner label.

For a given class y_i, consider the classification of x_i. If we have optimal classifiers at our disposal, for any $j \neq i$, all the $[ij]$ classifiers will vote for y_i. Therefore, class y_i will obtain all its $N-1$ votes. All other classes will obtain at most $N-2$ votes, but not more, and i should be elected the winner.

Algorithm

A set $\mathcal{B}$ of C_N^2 binary classifiers is trained for all the possible unordered pairs of classes in $\mathbb{Y}$. The binary classifier $[ab]$ should be trained with $\mathbb{X}_{train}^a \cup \mathbb{X}_{train}^b$, that is, training samples drawn from classes y_a and y_b.

Once all the classifiers have been trained, the run-time recognition is done according to algorithm 1.

A number of observations about this algorithm should be made. First, it is *a priori* suboptimal, because each binary classifier is trained independently one from another. Similarly, at run-time, the predictions of one classifier are irrelevant to other classifiers. An optimal multi-class system would try to optimise the decision rules for all the binary classifier jointly.

Another issue is that the assumptions made are optimistic:

- Because the classifiers are likely to be non-optimal, the legitimate class i may not win all its $N-1$ rounds.
- It is possible that the $[jk]$ classifiers "conspire" to all vote for a wrong class k. k could end up with more votes than i.

[1] $C_n^k = \binom{n}{k} = \frac{n!}{k!(n-k)!}$

[2] Optimal meaning here that the classifier never misclassifies.

Input: a sample $s \in \mathbb{X}_{test}$ to classify, $\mathcal{B}$, $\mathbb{Y}$
Output: a letter prediction
foreach $y \in \mathbb{Y}$ **do**
  $vote(y) = 0$
end
foreach *classifier* $[ab] \in \mathcal{B}$ **do**
  $prediction = [ab].Classify(s)$
  Increment $vote(prediction)$
end
$max\_nb\_votes = max_{y \in \mathbb{Y}}(vote(y))$
$winners = \{y \in \mathbb{Y} / vote(y) = max\_nb\_votes\}$
if *winners contains only one element* a **then**
  return a
end
if *winners contains only two elements* a, b **then**
  $w = [ab].Classify(s)$
  return w
end
else
  return a random element of $winners$
end

Algorithm 1: Majority voting with one-vs-one classifiers

- Another possibility is that no clear winner emerges, because there are two or more finalists after the voting session.

Note that when there are more than two finalists, ties are resolved randomly. The inability of such simple majority-voting systems to resolve n-ary ties is a known weakness. One way to solve this is to invoke a non-binary classifier of a radically different nature as a final arbiter, in order to select one letter over the remaining finalists[3].

Despite these weaknesses, one-versus-one classifiers remain interesting for two reasons. First, each binary classifier can specialise itself to focus on certain features of the observation vectors to make its predictions. When implementing the binary classifiers as SVMs with dynamic kernels, that means the score space can be tailored to the specific needs of each classifier. Also, no choice regarding the class ordering has to be made, so this scheme is robust in that respect. This is potentially not the case with the tree schemes, discussed in the next section.

[3] In our case, that system could be a standard HMM recogniser.

4.4 Decision graphs

A number of reductions have been devised based on directed binary graphs. A binary *directed acyclic graph* (DAG) is defined by a finite set of labelled nodes, and a finite set of directed arcs $i \rightarrow j$ between these nodes. The binary DAG is also subject to the following constraints:

- All nodes except one must have at least one parent node. The node with no parent is called the root node r.
- All non-leaf nodes must have exactly two children, $leftChild$, and $rightChild$.
- The graph is acyclic: there exists no sequence of nodes $(n_1, \ldots, n_i)$ so that n_{j+1} is a child of n_j, and $n_1 = n_i$.

Such graphs are typically trees (*i.e.* all non-root nodes have only one parent), but this is not necessarily the case, as in figure 4.2.

To use DAGs in pattern recognition, a class y must be associated to each leaf, and a binary classifier B assigned to each internal node. The recognition process is initiated by calling FollowGraph(s, r), which is described in algorithm 2.

Input: a sample s to classify, a graph node n
Output: a class prediction y
if *IsLeaf(n)* $= true$ **then**
  return $n.y$
end
if $n.B(s) = true$ **then**
  return FollowGraph(s, $n.leftChild$)
end
else
  return FollowGraph(s, $n.rightChild$)
end

Algorithm 2: FollowGraph

Decision directed acyclic graphs (DDAGs) are DAGs where the children of the i^{th} node in the j^{th} internal layer are the i^{th} and $(i+1)^{th}$ nodes in layer $(j+1)$. An example is given in figure 4.3.

A wide range of possibilities exists to implement the node classifiers. They could be one-versus-one, one-versus-the-rest classifiers, or they could classify a set of labels against another, *e.g.* classify $\{y_1, y_2, y_3\}$ as one set, and $\{y_4, y_5\}$ as the other. The divide-by-two algorithm [25] follows this idea. Other possibilities are described in [18], and [9], amongst others.

The DAGSVM, based on [17], is a DDAG with C_N^2 SVM classifiers. As shown in figure 4.3, each node is essentially a i-versus-j classifier. For a

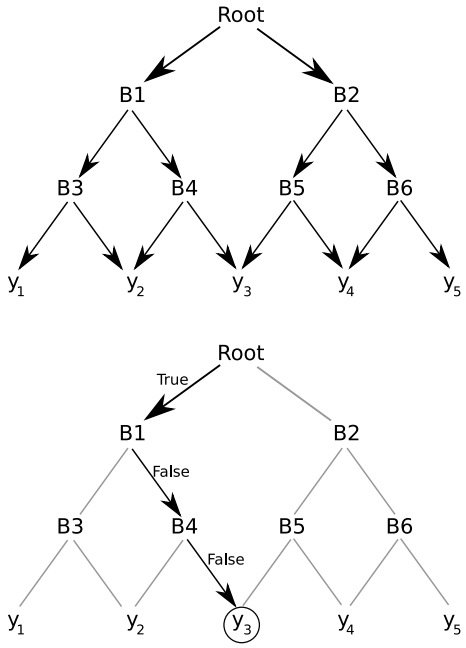

Figure 4.2: A directed acyclic graph, and going through it to classify a sample

given sample, if the $[ij]$ classifier votes for j, then the sample is decided not to be a i. In effect, each test removes a candidate class from the list of all possible classes. After $N-1$ tests, the sample is assigned to the remaining class.

By merging redundant nodes together (*i.e.* nodes which correspond to the same list of remaining candidates), the DAGSVM can be seen as an efficient generalisation of decision trees.

Although the number of classifiers is the same as with the one-versus-one majority voting algorithm, it is typically faster at run-time, because the number of evaluations is only in $N-1$, the height of the DAGSVM graph.

Platt reports that the DAGSVM relies on an arbitrary choice for the class order of the leaves, but experimentation with different orders did not yield significant changes.

4.5 Error-correcting output codes

Error-Correcting Output Codes (ECOC) are introduced by Dietterich & Bakiri in [6]. ECOC classification draws on the analogy of transmitting information (the class of the sample) over an unreliable channel. Bit-inverting errors (0 switched to 1, and vice-versa) occur because notably of the non-optimal feature space used, the limited training data available, and imperfect

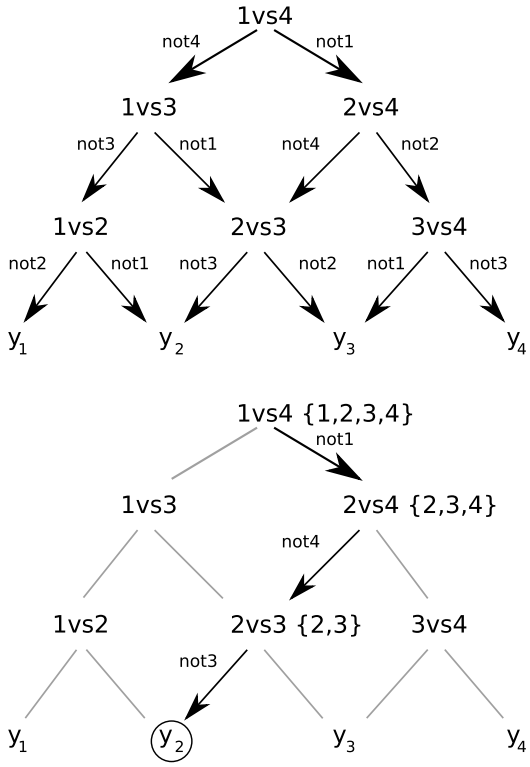

Figure 4.3: A DAGSVM

classifiers.

The ECOC algorithm tries to automatically correct transmission errors, as long as there are not too many of them. To do so, each class k is assigned a L-bit binary string c_k, called a *codeword*. Then L binary functions $f_l()$ are trained, one for each bit position of the codewords.

Class	Codeword				
	f_1	f_2	f_3	f_4	f_5
A	0	0	0	1	0
B	0	1	1	0	0
C	1	0	0	0	1
D	1	1	0	1	0

Table 4.1: An example 5-bit code table for a 4-classes problem. The codeword c_B for B is '01100'.

For a code length $L > log_2(k)$, there are 2^L possible codewords, of which only k are assigned. The classification of a sample s is done by first evaluating the L binary functions $f_l(s)$, and concatenating the L bits into a L-bit string c. Then, a lookup in the code table against c is performed. If c is a

valid codeword, then classifying it is done by returning the corresponding class. However, if c is not valid, that means at least one substitution error has occurred. The code must be compared against each of the valid codewords, and assigned to the class k whose code is the closest to c, according to some distance metric.

A useful distance metric over a set of fixed-length binary strings is the Hamming distance, which is defined as the number of bits that differs between two strings. The ability of an error-correcting code to detect and correct bits is directly linked to the minimum Hamming distance between any two rows of the code table. If a table has a minimum Hamming distance of $2n$, then the maximum number of bit substitution errors that the code can detect and correct is n.

As a consequence, the binary functions $f_l()$ must be chosen as to maximise the codeword Hamming distance, so that a larger number of errors can be corrected. The minimum Hamming distance of table 4.1 is $d(A, D) = 2$, so this table can correct only one substitution error.

It has been verified experimentally that functions defined as the presence or absence of "meaningful" features typically yield codes with a low Hamming distance, because the features end up being correlated to each other. Dietterich & Bakiri give an example for a hand-written digit recognition task, where a binary code built from six meaningful features ("the digit contains a vertical line", "the digit contains a closed curve", *etc.*) only has a Hamming distance of one.

Robust error-correcting codes must therefore be derived algorithmically using features without necessarily a high-level meaning. Unfortunately, the systematic construction of an optimal code for a given L and k is a non trivial problem. Different algorithms exist depending on the value of k. In this report, we concentrate on the case where $8 < k < 12$. For these values, no known algorithm to build the optimal code exists.

A simplified version of Dietterich & Bakiri's algorithm is proposed here. First, an exhaustive k-classes code of length $L = 2^{k-1} - 1$ is built as follows:

- Row 1 is all ones.
- Row 2 consists of 2^{k-2} zeros followed by $2^{k-2} - 1$ ones
- More generally, row i consists of an alternation of 2^{k-i} zeros and 2^{k-i} ones until the L bits are filled.

Since the first row is made entirely of ones, we suggest not to use it (*i.e.* not to assign it to a class), so that the classifier training is not biased towards class A.

For a digit recognition task, $k = 10$ so codewords are 511-bits long[4]. This represents too many classifiers, so we randomly extract a number of

[4] $k = 11$ if we include 'oh' as a synonym for 'zero'.

Class	Codeword						
	f_1	f_2	f_3	f_4	f_5	f_6	f_7
A	1	1	1	1	1	1	1
B	0	0	0	0	1	1	1
C	0	0	1	1	0	0	1
D	0	1	0	1	0	1	0

Table 4.2: The exhaustive 7-bit code for a 4-classes problem.

columns $L' < 511$, and compute the minimum Hamming distance of the obtained truncated code. By repeatedly performing this random selection process, we can select one that gave the largest observed Hamming distance. Since the distance between any pair of rows is expected to be binomially distributed with a mean of $k/2$, only extracted tables with a Hamming distance greater than $k/2$ should be considered.

4.6 Combining different types of classifiers

Pattern recognition is a difficult task, and classifiers do mistakes. However, because different algorithms tend to have different strengths and weaknesses, it is tempting to try to combine two sub-optimal classifiers into a hopefully more accurate one.

As we have discussed in section 4.3, it can happen that one-versus-one majority voting is unable to make a final decision, so it required another classifier as a fallback arbiter. In our case, it was simply a random choice, but it could have been replaced by a HMM classifier, or any other classifier.

It is possible to combine two systems in a more systematic way, provided that each classifier can assign a score, or some confidence measure, over the chosen classes. In the case of binary classifiers, the score can be a real number whose absolute value represents how sure the classifier is about the result, and whose sign represents one of the two classes.

Let B_1 and B_2 be two binary classifiers, and let $s_1()$ and $s_2()$ be their respective confidence score, defined so that samples with positive scores are assigned to class 1, and samples with negative scores to class 2. Given an arbitrary $\alpha \geq 0$, we can build an interpolated binary classifier I_{12}^{α} as follows:

- Let $s_{12}^{\alpha}(\vec{x}) = s_1(\vec{x}) + \alpha.s_2(\vec{x})$
- If $s_{12}^{\alpha}(\vec{x}) > 0$, assign $\vec{x}$ to class 1
- If $s_{12}^{\alpha}(\vec{x}) < 0$, assign $\vec{x}$ to class 2

If $\alpha = 0$, the interpolated classifier I_{12}^{0} is exactly equal to B_1. Similarly, $I_{12}^{+\infty}$ is equivalent to B_2. Let $a(\alpha)$ be the accuracy of I_{12}^{α}. We are interested

in maximising $a(\alpha)$ as a function of α. While this problem is typically not solvable analytically, by experimentally evaluating $a(\alpha)$ for different values of α, we can build an interpolated system that is guaranteed to be at least as accurate as B_1, or B_2, whichever is the best. Indeed, in the worst case where the accuracy of the interpolated system is always decreased, we can always fallback to $\alpha = 0$ or $\alpha = +\infty$.

When classifying two classes, the HMM log-likelihood ratio $log(P(\vec{x}|\mathcal{M}_1)) - log(P(\vec{x}|\mathcal{M}_2))$ is a natural choice for the score function. If a set of classes must be classified, say {1,2}-versus-{3,4}, then the following extended score should be used instead:

$$s_{12-34}^{HMM}(\vec{x}) = \log\left[\frac{\max\Big(P(\vec{x}|\mathcal{M}_1), P(\vec{x}|\mathcal{M}_2)\Big)}{\max\Big(P(\vec{x}|\mathcal{M}_3), P(\vec{x}|\mathcal{M}_4)\Big)}\right]$$

For SVM recognisers, we can use the algebraic distance $(\vec{\phi} \cdot \vec{x} + b)$ of $\vec{x}$ from the decision hyperplane. If the SVM is provided with a maximally non-committal metric, then all the dimensions of the feature space are considered to have the same importance. However, it is known that the HMM log-likelihood is a highly discriminative parameter. Hence, an interesting side-effect of the SVM-HMM interpolation is that the α parameter can be used to effectively scale the contribution of the HMM log-likelihood. A SVM-HMM interpolated recogniser has been built with encouraging results in [8].

More elaborate combination methods exist, such as boosting [19]. However, they will not be discussed in this report.

Chapter 5

Filter trees

In [2], Beygelzimer *et al.* present the filter tree algorithm, which reduces a multi-class classification problem into a binary classification. It effectively solves the cost-sensitive N-class classification problem by introducing *importance-weighted binary classifiers*, based on the idea that predicting some samples correctly is more important than for some others.

The filter tree can be seen as a kind of DAG (*cf.* section 4.4), although the training process is quite different. The filter tree is a binary tree T with N leaves, each of them being assigned to a distinct class.

5.1 Algorithm

Consider a cost-sensitive training set $(\mathbb{X}^{train}, cost(x, y_i))$, a binary filter tree T and an importance-weighted binary learner procedure `Learn`. The filter tree training is a bottom-up procedure, described in algorithm 3.

For each node n, the set $S_n = \{x, prediction, c\}$ is the training set for the importance-weighed learner $n.classifier$, c representing the importance of node n correctly classifying the sample x. The innovation of the filter tree training is that S_n depends on the nodes below n: samples that got misclassified by child nodes are ignored in the upper levels.

It is instructive to describe in detail how the training proceeds. Consider the simplified cost-insensitive problem where $\mathbb{Y} = \{a, b, c, d\}$, and where $\mathbb{X}$ is reduced to only one sample x_a, representative of class a. The cost of assigning x_a to a is zero, and the cost of assigning it to any of the three other classes is one.

The filter tree is fixed as shown in figure 5.1. Let us first assume that all the classifiers are optimal. Table 5.1 lists the content of S_n for each node of the tree. Because $card(\mathbb{X}) = 1$ in our simplified example, S_n is reduced to one sample as well.

Clearly it does not really matter how the sample x_a is classified by node $N2$, because x_a is neither a c nor a d. Hence, there should be no cost in

Input: X^{train}, $cost(x, y_i)$, T, `Learn`
Output: a trained filter tree

foreach *internal node* $n \in T$, *from the leaves up to the root* **do**
  $S_n = \emptyset$
  foreach *example* $x \in X^{train}$ **do**
    a = the prediction of the left subtree of n
    b = the prediction of the right subtree of n
    $c = |cost(x, a) - cost(x, b)|$
    $prediction = \text{argmin}(cost(x, a), cost(x, b))$
    $S_n = S_n \cup \{(x, prediction, c)\}$
  end
  $n.classifier = \texttt{Learn}(S_n)$
end
return T

Algorithm 3: Training a filter tree

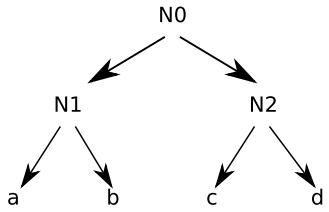

Figure 5.1: A four-classes filter tree example

misclassifying x_a at this point. Because the importance associated to such irrelevant samples is zero, they are effectively being ignored by the learning algorithm.

Now, if N1 happens to misclassify x_a, the sample will consequently be propagated upwards with its importance being set to zero, as shown in table 5.2, and therefore will be ignored by the upper nodes.

Testing is simply a standard top-down procedure that outputs the class y such that every classifier from the root to the leaf chooses y.

5.2 Advantages of the filter trees

In many practical pattern recognition problems, optimal solutions do not exist. It could be because the training set contains contradictory examples, or because the chosen feature space is not powerful enough to separate the samples into distinct classes. As a result, the absolute error rate is not that interesting when comparing classifiers. The more meaningful notion of *regret* is defined as the difference between a classifier's error rate on a given problem, and the smallest error rate possibly achievable for that problem. In

Node n	S_n
a	$\{(x_a, a, 1)\}$
b	$\{(x_a, b, 0)\}$
c	$\{(x_a, c, 0)\}$
d	$\{(x_a, d, 0)\}$
N1	$\{(x_a, a, 1)\}$
N2	$\{(x_a, ?, 0)\}$
N0	$\{(x_a, a, 1)\}$

Table 5.1: Training set S_n for all the nodes of the example filter tree

Node n	S_n
a	$\{(x_a, a, 1)\}$
b	$\{(x_a, b, 0)\}$
c	$\{(x_a, c, 0)\}$
d	$\{(x_a, d, 0)\}$
N1	$\{(x_a, b, 0)\}$!
N2	$\{(x_a, ?, 0)\}$
N0	$\{(x_a, ?, 0)\}$

Table 5.2: Training set S_n, if N1 misclassifies x_a

other words, regret quantifies the *avoidable* error rate for a given problem.

Table 5.3, adapted from [2], compares the different classifiers we have discussed in terms of the error rate, the regret, and the number of evaluations. In this table,

- e is the average binary error rate.
- r is the average binary regret.
- The 'Error' column gives an upper bound of the multi-class error rate as a function of N and e.
- The 'Regret' column gives an upper bound of the multi-class regret as a function of N and r. 'none' means that no regret transform provably exists.

Reduction	Error	Regret	Number of evaluations
1-vs-1	$(N-1)e$	$(N-1)r$	$\frac{1}{2}N(N-1)$
Tree	$\log_2(N)e$	none	$\log_2(N)$
ECOC	$4e$	none	$O(\log N)$
Filter tree	$\log_2(N)e$	$\log_2(N).r$	$\log_2(N)$

Table 5.3: Comparison of some reduction schemes (from [2])

Standard trees and filter trees are particularly efficient at run-time, because the number of evaluations varies in $O(\log_2 N)$. They are faster than the DAGSVM ($N-1$ evaluations), and much faster than the one-versus-one majority voting scheme (C_N^2).

The filter tree typically performs better than the other algorithms, except when compared to the ECOC: the filter tree error bound is not as good. However, Beygelzimer *et al.* prove that the filter tree is *consistent*, in the sense that given an optimal binary classifier (a classifier whose regret is 0), the reduction yields an optimal N-class classifier. It has been proven that the ECOC reduction (theorem 2 of [13]) and the tree reduction (section 5.1 of [2]) are inconsistent.

5.3 Application to speech recognition

Speech recognition systems must overcome two difficulties:

- Many applications, like dictation or car navigation systems, demand real-time transcriptions, so the ASR decoding must be fast enough. The limited number of evaluations of tree-based schemes makes the filter tree a suitable candidate.
- State-of-the-art systems cannot avoid errors, particularly in noisy environments. The good bound on regret of the filter tree gives us some guarantee that, provided we use good enough binary classifiers, like SVMs, the multi-class system will perform reasonably well.

This elicits the use of SVM-based filter trees in ASR systems. One way to do so is to proceed as follows:

- Given a vocabulary set, decide on a filter tree by choosing an arbitrary ordering of the words. Assign each leaf to a word.
- Train a SVM for each node of the tree, according to the cost-insensitive version of algorithm 3. Every sample that gets misclassified at any point is removed from the training set of upper SVMs.
- Run-time recognition is a standard top-bottom tree-walking procedure.

It should be noted that this scheme does not scale very well to large vocabularies. For a set of 2^n words, $2^{n+1}-1$ SVMs must be trained. A medium-sized vocabulary of 4096 words would thus require 8191 SVMs. The memory and CPU requirements for training that many SVMs can quickly become problematic. Filter trees should therefore be confined to only small vocabulary tasks. They remain particularly adapted to letter or digit recognition.

The question of which score space to use for each SVM remains open. Should every SVM use the same score space, or should they have their own? This question is answered through experimentations in chapter 7.3.

Chapter 6

Filter trees for noise-robust speech recognition

So far, the problem of noise has not been addressed. In this chapter, we briefly introduce the notion of model compensation, and how it can be applied to build a noise-robust filter tree based on SVMs.

6.1 Noise compensation schemes

Automatic speech recognition systems are typically trained in ideal situations, with perfect recording hardware, and negligible background noise. However, in real-life situations, the background noise cannot be controlled, and therefore modifies the audio recordings significantly. This introduces a mismatch between the training and the test conditions, which severely damages the accuracy of ASR systems.

To counter this mismatch, a wide range of adaptation schemes has been devised, like the Maximum Likelihood Linear Regression (MLLR) speaker adaptation [14], Cepstral Mean Normalisation [28] front-end processing, and uncertainty decoding [1], to name but a few.

Model compensation is a genre of adaptation which consists in training a model from clean training data, and then adapting the obtained clean model to a given noise environment by using noisy samples to re-estimate the model parameters.

6.2 Single-pass retraining

A simple model-based compensation scheme called Single-Pass Retraining (SPR) [7] is presented here. Given a specific noisy environment, assume

that stereo data is available[1] in the form of $(\mathbb{X}, \mathbb{C})$ where $\mathbb{X} = (\vec{x}_1, \ldots, \vec{x}_T)$ is the observed, noise-corrupted data, and $\mathbb{C} = (\vec{c}_1, \ldots, \vec{c}_T)$ is the set of the corresponding clean speech data. Let $\mathcal{M}_C$ be a HMM adequately trained on $\mathbb{C}$.

The SPR algorithm estimates the model parameters for a noise-adapted model $\mathcal{M}_X$ from the observed data $\mathbb{X}$ and the clean model $\mathcal{M}_C$, in a single pass. For each component m, the expected observed value $\vec{\mu}_m$ is approximated as:

$$\vec{\mu}_m \approx \frac{\sum_{t=1}^{T} \gamma_m^C(t) \vec{x}_t}{\sum_{t=1}^{T} \gamma_m^C(t)}$$

where $\gamma_m^C(t)$ is the posterior probability of the component m of the clean model $\mathcal{M}_C$ generating the observed vector $\vec{x}_t$ at time t. Similarly, the variance can be approximated as:

$$\mathbf{\Sigma}_m \approx \frac{\sum_{t=1}^{T} \gamma_m^C(t) (\vec{x}_t - \vec{\mu}_m)(\vec{x}_t - \vec{\mu}_m)^\top}{\sum_{t=1}^{T} \gamma_m^C(t)}$$

Compared to the MLE training (section 2.2), the transition matrix and the Gaussian component weights are not updated. Despite the simplicity of the algorithm, Gales ([7], [8]) reports nice improvements over non-compensated models.

6.3 Noise-dependent kernels and noise-independent SVMs

Having chosen a model adaptation scheme such as SPR, noise-adapted HMMs can be used to define a generative kernel as described in section 3.3.1. This gives us a sound framework for building a "noise-agnostic" filter tree. Assume that a set $(N_1, \ldots, N_n)$ of noise conditions have been defined. Let $\mathbb{C}$ be a set of clean data, and $(\mathbb{X}_1, \ldots \mathbb{X}_n)$ be the n associated sets of noise-corrupted samples, where each sample $x_i \in \mathbb{X}_i$ is the result of artificially adding noise from source N_i to the clean sample c_i. A noise-independent filter tree can be trained according to the following process:

- Train a set of HMMs with clean data from $\mathbb{C}$.
- Optionally split the noise conditions into a training set and a test set of noise conditions. This is useful to experimentally verify that the SVM-based filter tree is truly noise-independent.

[1] This is the case notably when noise is artificially added over clean recordings, as in the AURORA[15] task.

- For each training noise condition, estimate the noise-adapted generative model $\mathcal{M}_{N_i}$, and obtain from $\mathbb{X}_i$ a set of scores in a score space derived from $\mathcal{M}_{N_i}$. Use these scores to train a noise-independent SVM.
- Build a filter tree where each node is a noise-independent SVM.

The justification of this scheme (represented in figure 6.1) is that the noise-dependent kernels should nullify the signal variability that is caused by the noise, so that the SVM nodes can work on a noise-independent feature space.

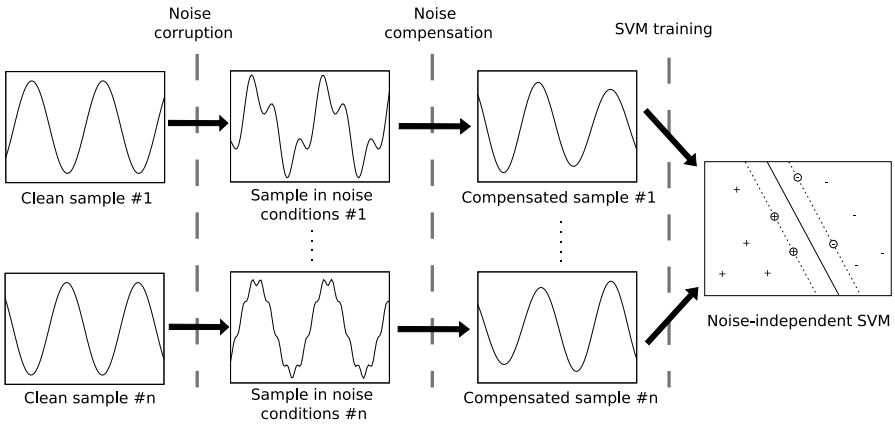

Figure 6.1: Training a noise-independent SVM

Chapter 7

The ISOLET task

7.1 The ISOLET database

7.1.1 Description

The speech database used for the first part of our experiments is the ISOLET spoken letter database, from the Oregon Graduate Institute.

The database contains two utterances for each of the twenty six letters of the alphabet, from 150 speakers, for a total of 7800 samples. The speakers were raised in various locations in the United States, and their age varies from 14 to 72 years.

The samples are grouped into five sets. Each set contains the 1560 utterances of 15 male speakers, and 15 female speakers. We will use the first fourth sets as our training data, and the last set as our evaluation data.

The recordings were sampled at 16000 Hz. Each sample is composed of a short silence (no longer than 50 ms), then the letter utterance, which lasts around 320 ms, and then another short silence. The recordings are clean, the mean SNR is 31.5 dB.

The waveforms are stored as OGI files. The OGI format is basically an uncompressed format with a metadata header. The HTK toolkit [30] natively supports this format. More details about the recording and the signal preprocessing can be found in [4].

7.1.2 The E set

The ISOLET database contains recordings for the 26 letters of the alphabet, but in this report, only the so-called "harder E set", or simply put, the "E set", is focused on. It is defined as the set of letters whose pronunciations end with the sound /*e*/, and thus comprises the following letters: `B`, `C`, `D`, `E`, `G`, `P`, `T`, `V`, `Z`.

There are two reasons why we want to focus on this reduced set of letters. First, Smith[23] reports that letters from the E set tend to be harder to

classify:

> For example, for the HMM system with 4 components per state, 74% of false classifications were for test samples drawn from the E set. The test set probability of error given an E-set letter was 7.2%, whereas given a non E-set letter, it was 1.4%.

Therefore, focusing on the most problematic letters will make differences in performances stand out more clearly.

The second reason is that this project required many various systems to be built and tested. It was thus interesting to reduce the number of letters considered. In particular, recognising the full alphabet with a one-versus-one majority-voting classifier requires training $C_{26}^2 = 325$ binary classifiers. If we restrict ourselves to the E set, the majority voting only requires $C_9^2 = 36$ classifiers, and the filter tree, 8 classifiers.

From now on, only the samples from the E set will be classified, and they will be classified against the 9 letters of this set, not against the 26 letters of the entire alphabet.

The following sections describe three baseline systems that provide a base for evaluating the filter tree algorithm.

Beforehand, some notation conventions are introduced. The full alphabet will be written as $\mathcal{A}$. The E-set will be referred to as $\mathcal{E}$, and its complement in $\mathcal{A}$, $\neg\mathcal{E}$.

The set of all sample data will be noted $\mathcal{S}$, and the subset of sample data drawn from $\mathcal{E}$, $\mathcal{S}^{\mathcal{E}}$. Where relevant, the distinction will be made between the training data $\mathcal{S}^{\mathcal{E}}_{train}$, and the test data $\mathcal{S}^{\mathcal{E}}_{test}$.

7.2 Baseline systems

7.2.1 HMM baseline classifier

First, we built a HMM recogniser using the Hidden Markov Model Toolkit, HTK[30], developed by the Cambridge University Engineering Department. Version 3.4 of HTK was used, which is the current stable release at the time of this writing.

Even though $\mathcal{E}$ is comprised of only 9 letters, we nevertheless trained 26 HMMs, one for each letter of $\mathcal{A}$. Doing so sharpens up the recognition accuracy on $\mathcal{E}$. Samples drawn from $\neg\mathcal{E}$ give information about the acoustic characteristics of $\mathcal{E}$, thus refining the extent of each HMM Gaussian component. We also built a HMM for modelling silence, which yields a total of 27 HMMs.

As represented in figure 7.1, the HMM initial template used for letters has 10 emitting states, with a left-to-right topology and self loops, but no

skips. Each state starts with a single-component, diagonal-covariance Gaussian distribution. The silence HMM template is similar, except that it has only one emitting state.

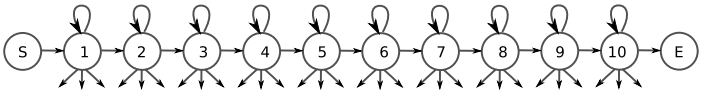

Figure 7.1: The HMM topology used

The observation feature space is based on 12 MFCC coefficients, plus the signal energy, plus the deltas and accelerations of these values, giving a total number of 39 `MFCC_E_D_A` coefficients. The conversion from the original waveforms to `mfc` files directly usable by the HTK toolkit was performed by the `HCompV` utility. The MFCC coefficients were based on a 20-channels filterbank. A Hamming windowing function was applied, and the analysis window size was 25.6 milliseconds long. The signal was pre-emphasised by a factor of 0.97. The other parameters[1] have been left to their default value. Appendix A.2 gives the full configuration file used.

HMMs were trained collectively using a maximum likelihood estimation scheme, as described section 2.2. The whole process was very closely based on Smith's thesis (*cf.* section 6.2 of [23]), and was as follows:

- compute the initial means and variances for single-component GMMs
- perform 20 iterations of embedded training
- split all emitting states into 2-components GMMs
- perform another 20 iterations of embedded training
- split all emitting states into 4-components GMMs
- perform another 20 iterations of embedded training

The initial model means and variances were computed using `HCompV`, by considering the relevant samples $\mathcal{S}^L_{train}$ for each letter L. A variance floor macro of 1% of the global variance was also inserted in the models, to compensate for the limited amount of training data available.

Embedded training was done with `HERest`. A mixture weight floor (`-w` option) of 1.1∗`MINMIX` was used. The number of Gaussian components was gradually increased from 1 to 4, using `HHEd`.

The training process did not try to prevent overtraining. In fact, looking at the evolution of the accuracy obtained with the various iterations, it was noted that the final HMM set was not the optimal one. However, two

[1]The exhaustive list of `HPARM` options can be found in table 5.4, section 5.18 of [28]

points should be made. First of all, the accuracy differences between the latter iterations were small, around ±1% absolute. Secondly, we are more interested in a systematic construction scheme. We will build other HMMs that way, so comparing them on a stable basis is crucial.

The HMMs obtained were used to build a baseline recogniser for the ISOLET database. Hypotheses were constrained to a silence-letter-silence grammar. The recognition was done with `HVite`, which implements a token-passing Viterbi algorithm, and returns the most likely hypotheses. `HResults` was used to aggregate statistics about the results.

The baseline performances on the training and the test data are summarised in table 7.1. Note that silences are not mentioned in the results, because they were always recognised correctly, due notably to the enforcement of the grammar network. Besides, the task of distinguishing silences from (clean) speech is a trivial[2] one. Including the silences as part of the recognition rate would artificially increase the accuracy, and therefore silences were systematically excluded.

Training data	2.27%
Test data	6.67%

Table 7.1: LER over $\mathcal{E}$ for the baseline HMM recogniser

Table A.2 in appendix A.4 gives the confusion matrix for the system. The letter error rate (LER) of 6.67% is the same as reported by Smith.

7.2.2 One-versus-one majority voting classifier

The second baseline system is a SVM-based, one-versus-one majority-voting classifier, based on section 4.3.

First, we trained a set $\mathcal{B}$ of $C_9^2 = 36$ SVM classifiers for all the possible unordered pairs of letters of $\mathcal{E}$: `B` vs `C`, `B` vs `D`, up to `B` vs `Z`, then `C` vs `D`, and so on.

It has been verified experimentally that the number of conflictual ties during the voting, *i.e.* when there were strictly more than 2 finalists, was limited to 3 or 4 occurrences out of 540, so the majority voting algorithm 1 rarely resorted to randomly picking a finalist. This confirms *a posteriori* that the crudely implemented resolution scheme was enough for our needs.

This algorithm was tested against many different score spaces. Here, results are given for only three of them. The first one (`1v1-FK`) was a Fisher kernel based on the log likelihood of a single HMM trained with all the samples of $\mathcal{S}^{\mathcal{E}}_{train}$. We call such an HMM a *merged* HMM, and refer to it as the `BCDEGPTVZ` HMM. The model template used for the training was initially

[2] Of course, in a noisy environment, separating "silence" from utterances is far from trivial.

the same as for single letter models, even though the number of Gaussian components was probably insufficient.

The score space was derived from the `mw` features of the Fisher kernel (*cf.* table 3.1 in section 3.3.4 for the meaning of this code).

The second setup (`1v1-GK`) used specific generative score spaces for each classifier. For instance, the SVM classifying `B`-vs-`C` was trained with samples from $\mathcal{S}^B_{train} \cup \mathcal{S}^C_{train}$. However, only `mw` score features were derived, so only derivatives for the first model `B` (with respect to its means and component weights) were used. A third setup (`1v1-GK-2`) was tried with `mwxz` score features to use both models.

All the score spaces were normalised using a Mahalanobis distance based on the inverse variance matrix.

Table 7.2 lists the results. Other score spaces not mentioned gave similar results, with an accuracy between 92% and 93.3%. It is striking to note that the majority voting is not doing better than a plain HMM recogniser, despite the fact that the 1v1 classifier is a more expensive scheme. In particular, generative kernels consume much more CPU, both for the off-line training, and the run-time classification, but in the end, the cost is not worth it.

Setup	Score features	Test LER	Correct predictions
`HMM`	N/A	6.7%	504 / 540
`1v1-FK`	`mw`	6.9%	503 / 540
`1v1-GK`	`mw`	7.8%	498 / 540
`1v1-GK-2`	`mwxz`	6.7%	504 / 540

Table 7.2: Error rates of the 1v1 classifier on $\mathcal{S}^{\mathcal{E}}_{test}$ for a few score spaces

Sifting though the logs, it was observed that a huge majority of the elected letters tend to win all their rounds, including wrongly classified letters. Table 7.3 displays the numbers of samples that won a certain number of votes, for the `1v1-FK` setup.

Number of votes won	Correctly classified	Incorrectly classified
8	499	37
7	4	0
1 to 6	0	0

Table 7.3: Distribution of the number of samples by their votes on `1v1-FK`

Winning the eight rounds of voting is not a guarantee that the result will actually be correct, even though we would have expected the eight different classification passes to complement each other, and constitute a more reliable system. In fact, only in 4 instances did the system adequately resolved (partially) inconsistent decisions, when 1 classifier out of the 8 was wrong.

It looks as if the majority voting scheme is not helping, because the outputs of the 8 classifiers $[ab]$ (a being fixed, $b \neq a$), are too correlated, and do not provide independent and complementary information.

The results obtained in this section are slightly worse than the one Smith reported in section 6.3 of [23], even though the experiments done here faithfully followed his description in section 5.4. His best system has a letter error rate of 5.4%. A possible explanation is that he used a customised version of SVM^{light}. Nevertheless, this is not a serious issue because our objective is not to try to build the most accurate letter recogniser, but to build instead a set of baseline systems to evaluate the usefulness of the filter tree approach for speech recognition.

7.2.3 ECOC classifier

If we want to compare the filter tree with the ECOC approach, we need to decide which length the error-correcting code should have. What should be the leading criterion for the comparison? Possible choices include:

- Best performance for each system. Increasing L widens the distance between the codewords, and therefore is expected to improve the performance. However, deciding how long L should be is unclear, and for scalability reasons, L cannot be very large. Above 20, the CPU requirements become intractable.
- Same number of binary decisions per samples, *i.e.* similar run-time cost. Anticipating a bit section 7.3, the ISOLET filter tree will have 4 levels. An ECOC with $L = 4$ means a mere $2^4 = 16$ different codewords. With such a short length, it is not possible to derive a 9-classes code with a minimum Hamming distance greater than 1. That means that the ECOC would be unable to correct any error.
- Same number of total binary classifiers to train. There are 1+1+2+4 = 8 binary classifiers in the filter tree, so we could take $L = 8$. That means 256 different codewords, and an optimal Hamming distance of 4.

As a compromise, the decision was made to derive a 16-bit code for the 9 classes, to favour the accuracy while keeping the complexity at an acceptable level. Following 4.5, a 16-bit code was generated, and is explicited in appendix A.3. Its Hamming distance is 6, hence the ECOC algorithm is able to correct 3 errors.

The practical application of the ECOC approach to the ISOLET task is now described. Because each class is a letter, each column of the code table yields a set of letter. With the chosen code, column one represents the string `DEGV`. The procedure for training the corresponding binary classifier is as follows:

1. Train a merged `DEGV` HMM, and the complementary `BCPTZ` HMM. The HMMs were trained in parallel with single HMMs for each of the 17 letters of $\neg\mathcal{E}$, as was done in 7.2.1.

2. Score the samples against the two HMMs. The best results were obtained with a node-specific `mwxz` score space.

3. Train a `DEGV-BCPTZ` SVM.

The classifiers corresponding to the remaining columns are trained in a similar fashion. The obtained error rate for the ECOC setup is given in table 7.4, along with the HMM baseline classifier.

Baseline system	Letter error rate
`HMM`	6.67%
`ECOC-GK`	6.30%

Table 7.4: LER for the ECOC system on $\mathcal{S}^{\mathcal{E}}_{test}$

While ECOC is doing better than the HMM baseline, the results were not as good as expected, at least with respect to the CPU cost of the method. In particular, training all the HMMs and the SVMs required by the ECOC algorithm was especially time-consuming (more than 20 times slower than for the HMM recogniser).

7.3 Evaluation of the filter tree algorithm

7.3.1 Direct implementation

A SVM filter tree recogniser, based on algorithm 3, was implemented to solve the ISOLET task. Since the way the binary tree should be ordered is not mentioned in the filter tree approach, we initially picked the natural alphabetical order. Figure 7.2 shows the resulting binary tree.

Amongst the variety of possible setups, the following two were chosen:

- FT-FK: Each SVM node is trained using the same `BCDEGPTVZ` Fisher kernel. The derived features are `mw`.

- FT-GK: Each SVM node is trained with a specific generative kernel. For instance, to build the node `BC-DE`, we train two merged HMMs, `BC` and `DE` with the samples from $\mathcal{S}^B \cup \mathcal{S}^C$, and $\mathcal{S}^D \cup \mathcal{S}^E$ respectively. The derived features are `mwxz` to use explicitly both the left and right models.

Again, the score spaces were normalised using the Mahalanobis distance, based on $\mathbf{\Sigma}^{-1}$.

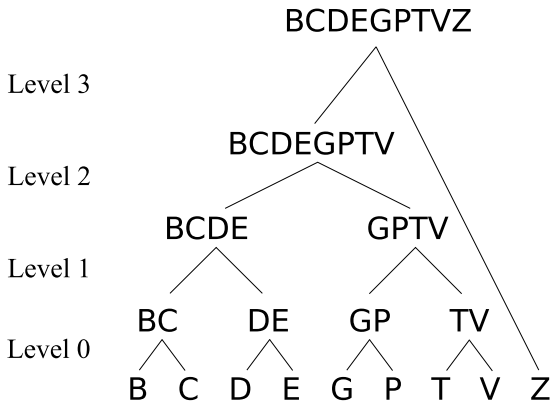

Figure 7.2: E-set filter tree, using the alphabetical order

Setup	Score features	LER	Correct predictions
HMM	N/A	6.7%	504 / 540
ECOC-GK	mwxz	6.3%	506 / 540
FT-FK	mw	8.1%	496 / 540
FT-GK	mwxz	7.4%	500 / 540

Table 7.5: Accuracy of the filter tree on $\mathcal{S}^{\mathcal{E}}_{test}$

The results are presented in table 7.5. The performances were not thrilling: the filter tree is doing worse than the three baseline systems. It is instructive to investigate what is happening, at each level (depth) of the tree. For now on, the following naming convention will be used:

- Level 0: refers to the 4 nodes classifying the leafs of the trees: B-vs-C, D-vs-E, G-vs-P, T-vs-V
- Level 1: refers to the 2 nodes classifying level 1 nodes: BC-vs-DE and GP-vs-TV
- Level 2: refers to the node classifying level 2 nodes, *i.e.* BCDE-vs-GPTV
- Level 3: refers to the root node: BCDEGPTV-vs-Z

We would like the filter tree to give a better accuracy than the plain HMM recogniser. We formulate this objective as a lower bound for the accuracy the filter tree should attain for each level. Let x be the average success rate for a set of SVMs, as used by the FT. There are 4 binary decisions to take to classify a sample from $\mathcal{E}$. Recall that the HMM baseline performance was 93.3%. In order to obtain an overall success rate better than, say, 94%, it would be sufficient to have the success rate x_p defined by:

$$x_p = \sqrt[4]{0.940} = 98.5\%$$

Note that x_p is a pessimistic estimate. The computation above assumes that the sets $\mathcal{F}_n$ of samples that get misclassified at level n are disjoint, in which case the errors are always cumulative. However, there is no reason *a priori* for this to be true. Mathematically, the assumption is:

$$\bigcap_n \mathcal{F}_n = \emptyset$$

However, if there exists in practical a sample $s \in \mathcal{F}_1 \cap \mathcal{F}_2$, then the error should be counted once, and not twice.

The relative good performances we obtained with the baseline HMM recogniser and the generative kernels used in one-versus-one majority voting are encouraging. That means the training process for level 0 classifiers is probably appropriate. However, levels 1 to 3 are not performing well enough. The performance of each level was therefore examined for the two setups. Table 7.6 gives for each setup the local accuracy score per level. The accuracy of level 1, for instance, is defined as the average accuracy of the `BC-DE` SVM over $\mathcal{S}_{test}^{BCDE}$, and of the `GP-TV` SVM over $\mathcal{S}_{test}^{GPTV}$.

Setup	Level	Test error rate	Correct predictions
`FT-FK`	0	1.7%	472 / 480
`FT-FK`	1	*3.1%*	465 / 480
`FT-FK`	2	*3.1%*	465 / 480
`FT-FK`	3	2.0%	529 / 540
`FT-GK`	0	1.0%	475 / 480
`FT-GK`	1	*3.1%*	465 / 480
`FT-GK`	2	*3.5%*	463 / 480
`FT-GK`	3	0.9%	535 / 540

Table 7.6: Accuracy level-by-level for the filter trees

Ideally, each level should have an accuracy of at least 98.5%. That means less than 2 errors per 120 samples, per SVM. Only the SVMs at levels 0 and 3 pass this criterion for `FT-GK`, or are close enough for `FT-FK`. This is an identified weakness in our direct implementation of the filter tree.

7.3.2 Increasing the model complexity

To address the problem of the naive filter tree's poor performance, the complexity of the models should be increased. Two methods were investigated:

- Strengthen the HMM models, by increasing the number of Gaussian components per state
- Increase the dimensionality of the score space

The HMM training scheme used seemed to be inappropriate for levels 1 and 2. This is actually not that surprising. We are using the same HMM template whether we are recognising single letters, or sets of letters. The number of four Gaussian components per state is adequate for level 0 nodes, but fails to capture the acoustic richness of merged sets in upper levels.

To explore this idea, experiments were done with more complex HMM templates for upper levels. For instance, level 1 HMMs were trained with 8 components per emitting state. To counter the growing problem of data sparsity, states were tied. Since by definition, all the letters of the E set end with the sound /*e*/, it made sense to tie the states at the end of each model. However, the results obtained were disappointing, and led to no decisive conclusions.

7.3.3 Concatenating scores

Results from the experiments so far indicate that building merged HMMs is not the correct approach for the filter tree. Since level 0 SVMs work pretty well, it is tempting to consider using them to build upper-level SVMs.

In order to build on the information of each SVM, the score concatenation scheme described in 3.3.3 was used.

As explained by table 3.1, deriving `mw` scores from each single-letter HMM yields a 1600-dimensional vector. Concatenating two of such vectors in level 0 produces a 3200-dimensional vector; four in level 1, a 6400-dimensional vector. If we blindly applied the same idea up to the level 3, we would end up with 14400 dimensions. Clearly, a score space feature selection is required.

According to section 3.3.2, only features with the largest Fisher ratios should be retained. To estimate the optimal value for the number of features to retain, the `BC`-vs-`DE` SVM was examined. Using `mx` features, scores from the four `B`, `C`, `D`, `E` HMMs were concatenated, and then filtered. The results are given in table 7.7.

# of retained features	Error rate
500	2.50%
1000	**1.25%**
2000	2.08%
4000	2.08%
5000	2.08%
6400	**1.25%**

Table 7.7: Error rate of the `BC`-vs-`DE` SVM, on $\mathcal{S}_{test}^{BCDE}$, for different numbers of features

It is remarkable that it was possible to significantly reduce the dimensionality of the score space, from the unfiltered $4 * 1600 = 6400$ down to

1000 features, while keeping the same accuracy. Although this is not the case here, it also happened occasionally that reducing the dimensionality increased the accuracy, probably because it removes noisy features that were confusing the classifier. Smith confirms this in [21].

We consequently chose to retain 1000 features for level 1 SVMs. A similar experiment was done on the level 2 `BCDE-GPTV` SVM, to select the dimensionality of the upper level.

# of retained features	Training LER	Test LER
500	1.61	4.58
900	1.35	3.54
1000	1.15	**3.33**
1100	**1.04**	3.54
1150	1.20	3.33
1250	1.04	3.75
1500	0.99	3.96
2000	4.38	4.38

Table 7.8: Error rate of the `BCDE`-vs-`GPTV` SVM on $\mathcal{S}^{BCDEGPTV}$ for different numbers of features

It was decided that 1000 features were also a good choice for the level 2 SVM. Three new SVMs based on this new score space scheme were then built. The error rates obtained are listed in table 7.9.

SVM	LER
`BC-DE`	1.25%
`GP-TV`	2.08%
`BCDE-GPTV`	3.33%

Table 7.9: Error rates with concatenated score spaces on levels 1 and 2

The new scheme gave improved results for the problematic levels 1 and 2. However, performances were still not satisfactory.

7.3.4 Composite implementation

Smith & Gales[22] report that the HMM log likelihood ratio $log(P_1/P_2)$ (score feature `l`) is a powerful discriminative feature. However, experiments done on `lmw` score spaces, instead of `mw`, did not notably improve the performances, probably due to the score space whitening. As suggested in 4.6, a composite approach was tried by interpolating the SVMs and the HMMs.

On level 1, the following experiment was tried:

1. For all $s \in \mathcal{S}_{test}^{BCDE}$, recognise s using first a `B-C` HMM recogniser, then a `D-E` HMM recogniser. This yields two log likelihood scores $s_{HMM}^{BC} =$

$\log \max\{P_B, P_C\}$, $s^{DE}_{HMM} = \log \max\{P_D, P_E\}$ for each sample.

2. Similarly, for all $s \in \mathcal{S}^{GPTV}_{test}$, compute s^{GP}_{HMM} and s^{TV}_{HMM}.
3. For all $s \in \mathcal{S}^{BCDE}_{test}$, compute the SVM score s^{BC-DE}_{SVM} using the `BC-DE` SVM
4. For all $s \in \mathcal{S}^{GPTV}_{test}$, compute the SVM score s^{GP-TV}_{SVM} using the `GP-TV` SVM
5. For a given $\alpha \geq 0$, classify $s \in \mathcal{S}^{BCDE}_{test}$ based on the composite score:

$$\begin{aligned} s_{comp}(\alpha) &= s^{BC-DE}_{SVM} + \alpha . \log \frac{\max\{P_B, P_C\}}{\max\{P_D, P_E\}} \\ &= s^{BC-DE}_{SVM} + \alpha . (s^{BC}_{HMM} - s^{DE}_{HMM}) \end{aligned}$$

6. Similarly, classify each $s \in \mathcal{S}^{GPTV}_{test}$
7. Compute the average accuracy $a(\alpha)$ of the composite classifier over $\mathcal{S}^{BCDEGPTV}_{test}$
8. Iteratively determine the optimal value for α.

$\alpha = 0$ gives the SVM accuracy of 98.33%, and $\alpha = +\infty$ gives the HMM accuracy of 96.25%. Note that α is determined to optimise the accuracy on both the BCDE and the GPTV sets conjointly. Also, the optimisation process was done over the test set, not the training set. Because the accuracy of SVMs over training data is very high, and occasionally equal to 100%, this interpolation would not have shown anything if the optimisation had been performed over $\mathcal{S}_{train}$. It is expected that the bias we introduce towards the test set will be limited, because we only tune one parameter per level (two parameters in total).

The accuracy graph obtained for level 1 is shown in figure 7.3. Although the interpolation scheme did not yield a reduction of the error rate, it is still interesting to note that there are two local maxima of the accuracy, for $\alpha = 0$ and $\alpha \in [10.8, 19.6]$. Even though the merged HMM level 1 classifier is not as good as the SVM, it is still able to contribute constructively in the recognition.

The value $\alpha = 16$ was chosen to build two composite level 1 `BC-DE` and `GP-TV` classifiers. Comparing the confusion matrices of both the initial SVM and the composite level 1 classifiers (table 7.10) shows that they essentially have the same behaviour. Therefore, it was deemed unnecessary to implement a composite level 1 classifier in this experiment.

A similar experiment was conducted on level 2. Again, 1000 features were retained, using the Fisher ratio. The training yielded the results of table 7.4.

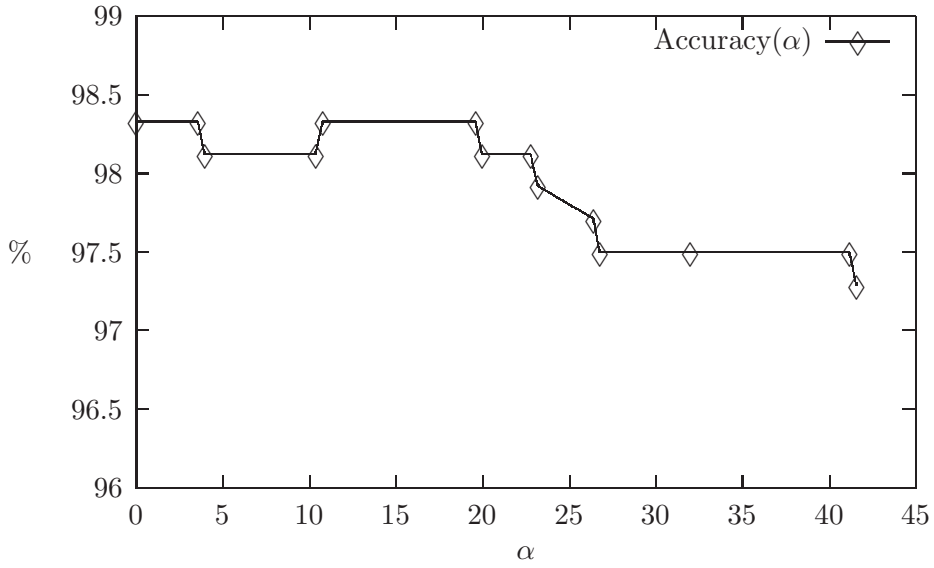

Figure 7.3: Accuracy of the level 1 composite recogniser as a function of α

Setup		BC	DE		GP	TV
SVM	BC	117	3	GP	116	4
	DE	0	120	TV	1	119
Composite	BC	118	2	GP	116	4
	DE	0	120	TV	2	118

Table 7.10: Confusion matrices for the composite and non-composite level 1 recognisers

A notable improvement over the SVM system was observed for $\alpha = 150$. The best error rate obtained on level 2 was 2.08%.

It is now worthwhile to redesign our filter tree algorithm to use what we have learnt so far. The second iteration of the system had the following characteristics:

- All the SVMs are based on concatenated score spaces with `mx` features[3].
- Level 2 SVM is a composite SVM-HMM recogniser configured with $\alpha = 150$.
- Level 1 is not composite.

[3]Because of a scalability issue in the internal tool that computes score statistics, only half of $\mathcal{S}_{train}$ was used to train the `BCDEGPTV-Z` SVM, in order to keep the memory usage below 2Gb. In the ISOLET database, each letter is pronounced twice by each speaker, so only the first utterance was used in level 3. All samples were used for the other levels.

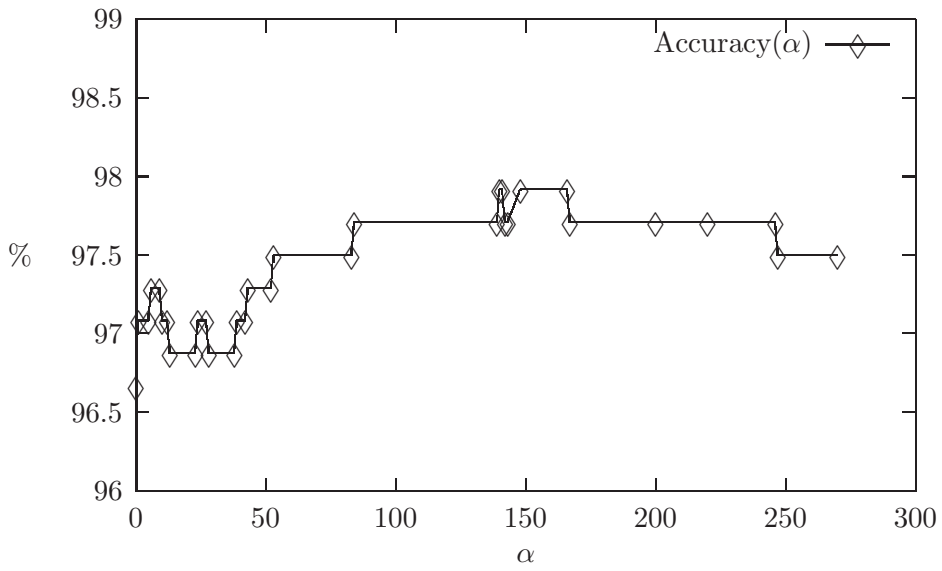

Figure 7.4: Accuracy of the interpolated level 2 recogniser as a function of α

Classifier	LER
SVM	3.33%
SVM/HMM	2.08%

Table 7.11: Accuracies for the composite and non-composite level 2 recognisers

- The alphabetical ordering is conserved (see figure 7.5).

With this configuration, the filter tree finally performed better: the final test accuracy was 94.26%, which is a +0.93% absolute improvement over the HMM baseline system (93.33%).

7.3.5 Robustness regarding the ordering

The filter tree paper[2] never mentions how the leafs of the tree should be ordered. It is only said that a binary tree T should be fixed, with no indication on how to do it. Is the filter tree actually robust against a reordering of its leafs? To find out whether this is the case, a number of different trees, shown in figure 7.5, were tried.

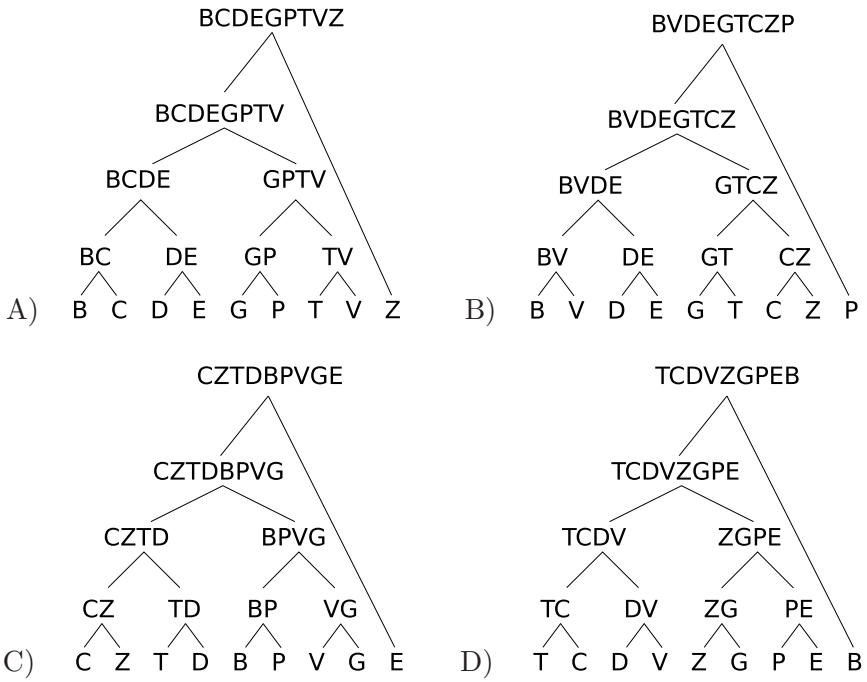

Figure 7.5: Various tree hierarchies

They were:

- A: The reference alphabetical ordering: `B`, `C`, `D`, `E`, `G`, `P`, `T`, `V`, `Z`
- B: An arbitrary ordering where leafs are paired in an attempt to regroup HMM confusable pairs.
- C, D: Two random orderings, generated by the computer

Levels 1, 2 and 3 were composite levels. The α parameter was set respectively to 16, 150 and 90. These values were obtained by optimising the performance of each level for the ordering A. The other orderings used the same α values, even though they might not have been the optimal ones.

Order	LER
A	5.37%
B	6.11%
C	5.37%
D	5.55%

Table 7.12: Error rates of the filter tree on various orderings

The results, listed in table 7.12, were encouraging. Not only has the accuracy globally improved, but it is also reasonably independent from the

choice of the binary tree. Moreover, even though the α were optimised for the setup A, they were reliable enough for the other setups.

It should however be noted that the insensitivity to ordering is actually not a consequence of the filter tree algorithm itself, but seems to be a characteristic of the composite nature of non-leaf SVMs. To prove this point, experiments have been conducted where composite SVMs were built only for level 2[4]. In such an environment, the error rates varied widely. Setup B in particular gave a disappointing error rate of 8.5%. Comparing tables 7.13 and 7.14 makes this fact more explicit. These two tables give the number of errors committed globally on each level, for each of the four setups. When all levels are composite (table 7.13), the number of errors at a specific level is relatively independent of the ordering. On the other hand, when only level 2 is composite (table 7.14), then the numbers of error on non-composite levels vary greatly. For instance, errors for level 1 go from 5 to 16.

Setup C is peculiar: it does many mistakes on level 0 in both setups, which is usually the most accurate level. On level 1 however, it does remarkably well.

Level → Order ↓	0	1	2	3
A	5	8	10	6
B	4	12	9	8
C	11	2	11	5
D	2	11	10	7
Avg LER (%)	99.0	98.5	98.1	98.8

Table 7.13: Number of errors by level, on various orderings, with composite levels. The four average LER per level are also given; three of them are greater that the target accuracy per level of 98.5%.

Level → Order ↓	0	1	2	3
A	5	8	10	6
B	4	16	9	16
C	10	5	10	6
D	2	13	10	9

Table 7.14: Number of errors by level, on various orderings, with only level 2 being composite

[4]Level 2 was selected for being the weakest one regarding measured accuracy.

7.4 Results table

Table 7.15 summarises the best results obtained on the E set with the three baseline systems and the filter tree.

System	Letter error rate
HMM	6.67%
1v1-GK-2	6.67%
ECOC-GK	6.30%
FT naive	7.40%
FT composite	5.37%

Table 7.15: Error rates on the ISOLET task for the E set

Chapter 8

The AURORA task

Having obtained interesting results on the ISOLET classification, it was deemed worthwhile to apply the filter tree on a more realistic task. The AURORA task is more challenging than the ISOLET task, notably because it involves noisy speech, and it is a continuous recognition task.

Due to the limited remaining time, not many experiments could be tempted on the AURORA task. However, as discussed below, the few results obtained were interesting.

8.1 The AURORA database

The AURORA database [15] has been designed to evaluate ASR systems in noisy conditions. The source speech is the TIdigits connected digits task, where sequences of up to 7 digits are uttered by male and female American speakers. The digits include 'oh' as a synonym for 'zero', giving a total of eleven allowed words. The recordings were sampled at 20kHz in clean conditions, then downsampled at 8kHz using a low-pass filter with a cutoff frequency of 4kHz.

Only a subset of the database has been used in this experiment. For that subset, stereo data was available so single-pass retraining, as discussed in section 6.2, was applicable.

Four types of noise were artificially added to the clean speech data. For the same type of noise, different signal-to-noise ratios were used to corrupt the clean speech: 20dB, 15dB, 10dB and 5dB. Hence, 16 noise conditions were defined. For now on, the `Ni_SNRjj` notation will be used to designate the data set obtained by adding noise `Ni` with the SNR `jj` dB on the clean speech.

For each noise condition, 422 training utterances and 1001 test utterances are available.

39-dimensional feature vectors, consisting of 12 MFCC coefficients including the zeroth cepstrum, as well as the deltas and accelerations, was

Code	Description
N1	suburban train
N2	babble from a crowd of people
N3	car
N4	exhibition hall

Table 8.1: The different type of noises added

extracted from the input data. Details of the feature extraction are given in appendix A.2.

HMMs were trained for each word with 16 emitting states and 3 Gaussian components per state. Silence and inter-word pauses models were trained as well, with respectively 3 and 1 emitting states, and 6 Gaussian components per state.

8.2 Implementation of the filter tree

The implementation was loosely based from [8], with the notable difference that the recognition algorithm was replaced by a filter tree. Building on section 6.3, the key objective here is to train a set of noise-independent SVMs that rely on noise-dependent kernels to adapt to the specific acoustic environment.

First, for each noise condition N, described in section 8.1, a set of HMMs $\{\mathcal{M}_N\}$ was trained using SPR, as was explained in 6.2.

For each training sample recorded in noise condition N, 11 feature vectors were derived using the $\{\mathcal{M}_N\}$ models. Only the components means were considered here (`m` features), and the vectors were limited to 1500 dimensions, using the Fisher criterion. The vectors were then concatenated, as was done in 7.3.3.

SVMs for the filter tree were trained using samples from nine sets out of sixteen: `N2_SNR05`, `N2_SNR10`, `N2_SNR15`, `N3_SNR05`, `N3_SNR10`, `N3_SNR15`, `N4_SNR05`, `N3_SNR10` and `N4_SNR15`. Not all the noise conditions were used for the training in order to verify whether the filter tree can be noise-independent.

The filter tree was tested on two noise conditions[1]: `N1_SNR20` and `N2_SNR05`, which are the sets with respectively the best and the worst error rate when using the HMM-SPR recogniser.

Compared to the ISOLET database where each sample was a single letter, here a sample represents a sequence of digits. Therefore, before we can use the filter tree, the utterance must be split into speech segments which should be aligned as closely as possible to the real uttered digits. This was

[1]Due to the limited time, it was not possible to test all the noise conditions.

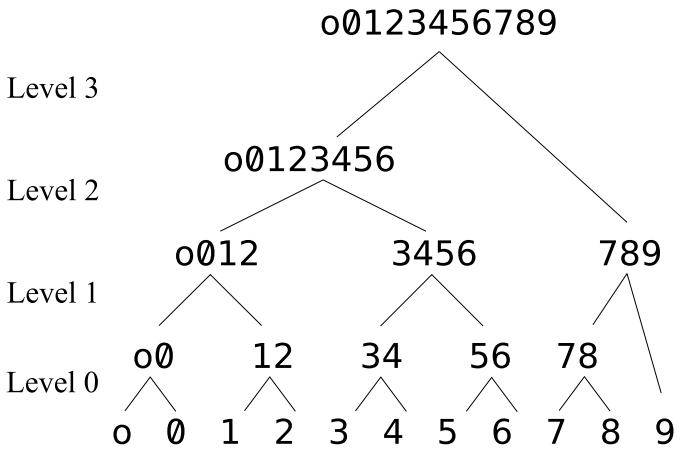

Figure 8.1: The binary tree used for the Aurora task

done using the SPR-adapted HMM recogniser. Then, each fragment of the utterance went through the filter-tree for digit identification.

The binary tree fixed for the task is shown in figure 8.1. To follow [8] more faithfully, an additional node should have been added to represent silences, but this was not done due to the limited time. The consequence of this is that the filter tree recogniser will not be able to correct deletion and insertion errors.

Each SVM level was interpolated with the associated HMM recogniser. The interpolation values used for each level were as follows: $\alpha_0 = 35$, $\alpha_1 = 30$, $\alpha_2 = 50$, $\alpha_3 = 90$.

8.3 Results

First, it was verified (table 8.2) that the SPR training is a remarkably effective adaptation scheme, and gave good results over unadapted HMMs. In particular on `N2_SNR05`, the unadapted HMMs are, as expected, completely unusable, while the SPR-adapted HMMs gave a respectable error rate of 18.83%.

Classifier	Noise condition	WER
Clean HMM	`N1_SNR20`	2.76%
HMM-SPR	`N1_SNR20`	1.60%
Clean HMM	`N2_SNR05`	73.82% !
HMM-SPR	`N2_SNR05`	18.83%

Table 8.2: Word error rates for unadapted and SPR-adapted HMMs

Table 8.3 compares the performance of the HMM-SPR and the composite filter tree classifiers. The filter tree was able to reduce the word error rate

Noise condition	Classifier	Number of errors			WER %
		Subs	Dels	Ins	
N1_SNR20	HMM-SPR	33	8	13	1.60%
N1_SNR20	FT-comp	25	8	13	1.41%
N2_SNR05	HMM-SPR	194	34	363	18.83%
N2_SNR05	FT-comp	180	34	363	17.44%

Table 8.3: Error rates on the AURORA task

in both cases. Insertion errors, more than substitutions, are the largest source of errors for N2_SNR05. Unfortunately, these stand uncorrected in our setup. Still, it is encouraging to see that filter tree was able to adapt to new noise conditions, so the combination of noise-dependent kernels with noise-independent SVMs was successful.

Gales's combined HMM/SVM system[8] has an error rate of 18.05% on the N2_SNR05 set. This figure does not include the gain induced by correcting insertions and deletions errors, so that the error rates between the two setups are comparable. While Gales used a crude interpolation policy (one α value, manually estimated on a small subset of data), experiments seem to indicate that the performances of both systems were relatively stable with respect to α.

The filter tree composite system therefore gave encouraging results on a realistic, noise-corrupted task. A caveat though: although Gales's results with SPR-trained HMM were exactly reproduced, it was unfortunately discovered very late that the MFCC encoding used for the composite filter tree setup was slightly different than for the adapted HMM. Re-training HMMs with the second MFCC encoding gave better results. The filter tree behaved slightly worse at first when the α factors were left unchanged. Relatively better results were obtained by re-estimating the α. The problem is that the accuracy was not robust against the choice of the α's. This is surprising because that was not the case for the ISOLET task. As a conclusion, the gain provided by the filter tree on the AURORA task is still unclear, and the results should be taken with a pinch of salt until more investigation is conducted.

System	Test set	Word error rate
HMM SPR	N1_SNR20	1.60%
HMM/SVM partial rescoring	N1_SNR20	1.38%
FT composite	N1_SNR20	1.41%
HMM SPR	N2_SNR05	18.83%
HMM/SVM partial rescoring	N2_SNR05	18.05%
FT composite	N2_SNR05	17.44%
HMM/SVM complete rescoring	N1_SNR20	1.38%
HMM/SVM complete rescoring	N2_SNR05	11.09%

Table 8.4: Best performance for each system. The HMM/SVM rescoring figures are taken from [8].

Chapter 9

Conclusions

In any decision graph where there is a unique path from the root to any leaf, it is enough to take only one wrong binary decision anywhere along the path to cause a global misclassification. For the filter tree on the ISOLET task, that represents four occasions of making a mistake, for each sample. *A contrario*, ECOC are inherently capable of correcting local errors, as long as they are limited in numbers. Therefore, it could have been expected that ECOC would perform better than the filter tree. However, that was not the case on the ISOLET task. Indeed, the filter tree, at least in its final form, gave significantly better results, while ECOC only did slightly better than the baseline HMM recogniser. This is all the more surprising that the filter tree is relatively faster to train and to use than the ECOC. It seems to indicate that the bound on the regret, as derived in [2] could be a relevant qualitative indicator of a classifier's expected performance.

This report proved that filter trees can successfully be applied to the domain of speech recognition. This fact was not immediately obvious since first, generative kernels had to be introduced, score spaces concatenated and filtered, and finally connected speech had to be segmented with a standard HMM recognizer. Moreover, the first implementation without SVM/HMM composition gave slightly disappointing results, and was sensitive to the ordering of its leafs. Two points are believed to be paramount for an efficient filter tree:

- The complexity of the generative score space must be carefully controlled. Even though the Fisher ratio might not be the best informational criterion, using it to select meaningful dimensions was quite effective.
- The HMM log likelihood ratio was a crucial piece of information, and required to be scaled appropriately. Using a composite HMM/SVM system was one way to achieve that.

On the other hand, the HMM training method, and score features other

than the log likelihood ratio and the derivatives with respect to the means (`l`, `m`), were comparatively less interesting.

The filter tree has shown its limits on a few occasions though. First, in the form used in this report, it remains restricted to small vocabulary tasks. It would be interesting to see if SVMs can be trained on phone speech units, without context, to solve LVCSR tasks. For 40 phones, that would mean a filter tree of height 7. It would also greatly rely on an adequate segmentation of the speech into phones from an upstream system. Triphones, as used in many HMM LVCSR systems, are not going to be tractable, because that would mean training around 23,000 SVMs for all the (commonly) observed triphones in the English language.

More intriguing was that the choice of the α's seemed robust for the ISOLET task, but not for the AURORA task. This could be a consequence of the presence of noise, but it is hard to be conclusive here.

More investigation is thus warranted. First, the questions of how the α parameter should be estimated, and how stable the derived filter tree accuracy is robust in that regard, remain open.

For the more ambitious AURORA task, the effectiveness of the filter tree to correct deletion and insertion errors by inserting a silence node in the tree should also be evaluated. It could also be very interesting to replace the SPR adaptation by a more sophisticated model compensation scheme, to see if the accuracy is improved, and if the choice for the α's becomes robust.

Finally, the comparison on the ISOLET task ended up being a bit unfair for the ECOC, because we greatly improved the score spaces for the filter tree, but not for the ECOC. It would be interesting to benchmark the filter tree and the ECOC approaches with more comparable score spaces.

Bibliography

[1] J.A. Arrowood, M.A. Clements, "*Using observation uncertainty in HMM decoding*", ICSLP '02, pp. 1561–1564, 2002

[2] A. Beygelzimer, J. Langford, P. Ravikumar, "*Multiclass Classification with Filter Trees*", 2007

[3] C.J.C. Burges, "*A tutorial on Support Vector Machines for Pattern Recognition*", Data Mining and Knowledge Discovery, Vol. 2, pp. 121–167, 1998

[4] R. Cole, Y. Muthusamy, M. Fanty, "*The ISOLET spoken letter database*", Technical report CSE 90-004, Oregon Graduate Institute of Science and Technology, 1994

[5] A.P. Dempster, N.M. Laird, D.B. Rubin, "*Maximum likelihood from incomplete data via the EM algorithm*", Journal of the Royal Statistical Society, Vol. 39, pp. 1–38, 1977

[6] T.G. Dietterich, G. Bakiri, "*Solving Multiclass Learning Problems via Error-Correcting Output Codes*", Journal of Artificial Intelligence Research, Vol. 2, pp. 263–286, 1995

[7] M.J.F. Gales, "*Model-based techniques for noise robust recognition*", Ph.D. thesis, 1995

[8] M.J.F. Gales, C. Longworth, "*Discriminative classifiers with generative kernels for noise robust ASR"*, working document, not published yet.

[9] T. Hastie, R. Tibshirani, "*Classification by pairwise coupling*", Annals of Statistics, Vol. 26, No. 2, pp. 451–471, 1998

[10] C. Hsu, C. Lin, "*A comparison of methods for multi-class support vector machines*", Technical report, Department of Computer Science and Information Engineering, National Taiwan University, 2001

[11] T. Jaakkola, D. Haussler, "*Exploiting generative models in discriminative classifiers*", Department of Computer Science, University of California, 1998

[12] T. Joachims, "*Making large-scale SVM Learning Practical*", in Advances in Kernel Methods - Support Vector Learning, B. Schlkopf, C. Burges, A. Smola (ed.), MIT-Press, 1999

[13] J. Langford, A. Beygelzimer, "*Sensitive Error Correcting Output Codes*", Conference on Learning Theory, pp. 158–172, 2005

[14] C. Leggetter, P.C. Woodland, "*Flexible Speaker Adaptation Using Maximum Likelihood Linear Regression*", Eurospeech 95, pp. 1155–1158, 1995

[15] D. Pearce, H.G. Hirsch, "*The AURORA experimental framework for the performance evaluations of speech recognition systems under noisy conditions*", ISCA ITRW ASR, 2000

[16] J. Platt, "*Fast training of support vector machines using sequential minimal optimization*", in Advances in kernel methods: support vector learning, ISBN 0-262-19416-3, 1999

[17] J. Platt, N. Cristianini, J. Shawe-Taylor, "*Large Margin DAGS for Multiclass Classification*", in Advances in Neural Information Processing Systems 12, pp. 547–553, 2000

[18] J.R. Quinlan, "*C4.5 Programs for Empirical Learning*", Morgan Kaufmann, ISBN 1-558-60238-0, 1993

[19] R. Schapire, "*The boosting approach to machine learning: An overview*", MSRI Workshop on Nonlinear Estimation and Classification, 2001

[20] N.D. Smith, M.J.F. Gales, M. Niranjan, "*Data-dependent kernels in SVM classification of speech patterns*", Technical Report CUED/F-INFENG/TR.387, 2001

[21] N.D. Smith, M.J.F. Gales, "*Using SVMs to classify variable length speech patterns*", Technical report CUED/F-INFENG/TR.412, Cambridge University Engineering Department, April 2002

[22] N.D. Smith, M.J.F. Gales, "*Speech recognition using SVMs*" in Advances in Neural Information Processing, MIT Press, 2002

[23] N.D. Smith, "*Using Augmented Statistical Models and Score Spaces for Classification*", Ph.D. thesis, 2003

[24] V.N. Vapnik, "*Statistical Learning Theory*", ISBN 0-471-03003-1, 1998

[25] V. Vural, J. Dy, "*A hierarchical method for multi-class support vector machines*", in Processings of the 21st International Conference on Machine Learning, 2004

[26] J. Weston, C. Watkins, "*Multi-class support vector machines*", Technical Report CSD-TR-98-04, Department of Computer Science, University of London, 1998

[27] Y. Xia, "*A new neural network for solving linear and quadratic programming problems*", IEEE Transactions on Neural Networks, Vol. 7, No. 6, pp. 1544–1548, Nov. 1996

[28] S.J. Young, G. Evermann, M.J.F. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J.J. Odell, D.G. Ollason, D. Povey, V. Valtchev & P.C. Woodland, "*The HTK Book (for version 3.4)*"

[29] Matthew Gerber's homepage: `http://links.cse.msu.edu:8000/members/matt_gerber/`

[30] The HTK homepage: `http://htk.eng.cam.ac.uk/`

[31] A SVM Java applet: `http://www.smartlab.dibe.unige.it/Files/sw/Applet%20SVM/svmapplet.html`

[32] SVM^{light}, a support vector machine toolkit by Thorsten Joachims: `http://www.cs.cornell.edu/people/tj/svm_light/`

Appendix A

Implementation details

A.1 SVM^{light}

The SVMs were implemented using Joachims's SVM^{light} toolkit[32], based on [12]. Training was done using `svm_learn` with the default options. In particular, linear kernels were always used, and the C trade-off parameter was left at its default value.

SVM classification over TCP

Gerber[29] has written a small extension for SVM^{light} that allows `svm_classify` to act like a TCP/IP server. This turned out to be quite handy for this project.

The filter tree approach requires each sample to go through many different SVMs. Because `svm_classify` takes a significant amount of time to parse its model file, it was not tractable to spawn a process `svm_classify` for each sample, and each SVM. It was also not practical to process all the samples through one SVM in one batch. A nice way to solve this practical hurdle was to spawn one TCP/IP server for each SVM classifier, and use these servers repeatedly through the filter tree training, and the classification. That way, each SVM were initialised only once, and the CPU cost was dramatically reduced.

The code of this extension was not used as is, but was modified to correct a critical bug, triggered by the large amount of data that was sent through the TCP tunnel.

A.2 MFC files generation

The following listing gives the configuration options specified to `HCopy` to generate the `mfc` files from the original ISOLET `ogi` files.

```
SOURCEFORMAT = OGI
SOURCEKIND = WAVEFORM
TARGETKIND = MFCC_E_D_A
NUMCEPS = 12
NUMCHANS = 20
TARGETRATE = 100000
WINDOWSIZE = 256000
```

The following listing gives the configuration options that was used for the AURORA task.

```
TARGETKIND      = MFCC_0_E
SOURCEFORMAT    = NOHEAD
TARGETRATE      = 100000.0
SAVECOMPRESSED  = T
SAVEWITHCRC     = T
WINDOWSIZE      = 250000.0
USEHAMMING      = T
ENORMALISE      = F
ZMEANSOURCE     = F
PREEMCOEF       = 0.97
NUMCHANS        = 24
CEPLIFTER       = 22
NUMCEPS         = 12
SOURCERATE      = 1250.0
LOFREQ          = 64
HIFREQ          = 4000
```

A.3 ECOC code

Table A.1 gives the 16-bit long ECOC code used in section 7.2.3. It can be noted that the various systems considered often have a couple of outstanding confusable pairs. However, these pairs change from a system to another. In particular, SVMs based on generative kernels always had different confusable pairs than the HMMs that were used to build the score space.

Class	Binary codeword															
B	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
C	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	1
D	1	1	1	0	0	0	1	0	0	1	1	1	1	0	0	0
E	1	0	0	1	1	1	1	0	1	0	0	1	1	0	0	0
G	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	1
P	0	0	0	0	1	1	0	1	0	1	0	0	1	1	0	0
T	0	1	0	0	0	1	1	1	0	0	1	1	0	1	0	0
V	1	0	1	0	1	0	0	1	0	0	0	0	1	0	0	1
Z	0	0	1	1	1	0	1	0	1	1	1	0	0	1	0	1

Table A.1: 16-bit long ECOC code used in the experiments

A.4 Confusion matrices

	B	C	D	E	G	P	T	V	Z
B	54			**4**				2	
C		58							2
D			55	1			1	3	
E	**4**		1	54		1			
G					57		3		
P						59		1	
T		1			2	2	55		
V	3			1				56	
Z		3						1	56

Table A.2: Confusion matrix on the E set for the baseline HMM recogniser

	B	C	D	E	G	P	T	V	Z
B	53		1	4				**2**	
C	1	59							
D			54	2	2		1	1	
E			2	57		1			
G					57		3		
P	1					57	1	1	
T					2	2	56		
V	**6**		3				1	50	
Z		4						2	54

Table A.3: Confusion matrix for the best majority voting system, `1v1-GK-2`. Notice the most confusable pair `B/V`. Compared to the baseline HMM system, the pair `B/E` is recognised more accurately here.

	B	C	D	E	G	P	T	V	Z
B	55	**5**							
C	**1**	59							
D			55			1	2	1	1
E	1			56			3		
G			1	2	56			1	
P			1		2	51		3	3
T			1	1			57	1	
V			1	1				58	
Z				1					59

Table A.4: Confusion matrix for the ECOC classifier.