DESIGN OF HIERARCHICAL CLASSIFIERS FOR EFFICIENT AND ACCURATE PATTERN CLASSIFICATION

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science (by Research)
in
Computer Science

by

M.N.S.S.K. Pavan Kumar
200399005
pavan@research.iiit.ac.in

International Institute of Information Technology
Hyderabad, INDIA
Aug. 2005

Abstract

Pattern recognition is an important area of research in information science with widespread applications in image analysis, language processing, data mining, bioinformatics, etc. Most of the research in this area was centered around building efficient classifiers for recognition problems involving two classes of samples. However, most of the real world pattern recognition problems are not simple binary problems; they have multiple classes to recognize. It is only in the recent past that researchers have started to focus on building classifiers for multiple class classification. There are two popular directions for developing multiple class classifiers — one is to extend the theory of binary classification directly to adopt to the multiple classes, and the other is to build large multiclass systems with efficient binary classifiers as components. The latter paradigm has received much attention and popularity due to its empirical success in various applications such as Optical Character Recognition, Biometrics, Data Mining etc. This Thesis focuses on the classifier combination paradigm to design efficient and accurate classifiers for multiclass recognition of large number of classes.

Decision Directed Acyclic Graph (DDAG) classifiers integrate a set of pairwise classifiers using a graph based architecture. Accuracy of the DDAG can be improved by appropriate design of the individual nodes. An optimal feature selection scheme based on Linear Discriminant Analysis (LDA) is employed for improving the performance of the nodes. This feature selection scheme extracts separate set of features for each node, which increases the time taken for a sample to be classified. A novel approach for the computation of a condensed representation of the large set of features using PCA is proposed to overcome this problem. Nodes with low performance are boosted using a popular boosting algorithm called Adaboost. Improvement in performance is demonstrated on Character Recognition and other datasets.

The arrangement of nodes in a DDAG is shown to affect the classification performance of the overall classifier. Popular DDAG algorithm is very accurate; however provides no more information than the class label. We modify this algorithm to handle a 'reject-class' for improving the performance. The design of this DDAG is posed as an optimization problem in terms of misclassification probabilities. The problem being NP-Hard, approximate algorithms are proposed to design an improved DDAG. It is experimentally shown that the proposed algorithm provides a design close to the average case performance of the DDAG.

Although DDAGs are attractive for building multiclass classifiers, they need large number of component classifiers to solve the problem. A Binary Hierarchical Classifier (BHC) is an alternate approach, which uses much fewer component classifiers compared to a DDAG. We propose a new scheme to divide the set of available classes into overlapping partitions so as to maximize the margin at each node of the BHC, and thereby improving the performance. BHC and DDAGs are the classifiers with complementary advantages. A DDAG has high accuracy but has high storage and classification time complexity. BHC is extremely efficient in storage, with a reasonably high accuracy. We exploit the advantages of both these classifiers to design a new hybrid classifier. This employs binary partitions at most of the nodes where the classification is relatively easy, and DDAGs for classifying complex subset of classes, wherever appropriate. The hybrid architecture is shown to perform better than the BHC, with a performance close to that of a DDAG. A great reduction in the number of nodes is achieved, compared to that of a complete DDAG.

This thesis presents a spectrum of classifier design algorithms for improving performance by feature selection, component classifier selection and combination architecture design. The design problems are formulated in their generality, and their complexity is analyzed. The proposed solutions are empirically evaluated on popular datasets. Experimental results demonstrate superior performance of the algorithms.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Classification of patterns emerging in real-life situations is an important problem in intelligent systems. Measurements from different processes results in conceptually different classes. Their automatic identification, generalization and thereafter derivation of an appropriate description are the fundamental issues of interest to any pattern classification system [19]. These techniques are extensively used in computer vision, speech and signal processing, data mining, Bioinformatics, weather prediction and many other walks of information sciences.

Pattern recognition algorithms build an appropriate description of the knowledge from large amount of labeled examples during a supervised training phase. This knowledge is later used in labeling an unseen example in an accurate manner. Accuracy of this labeling is an important measure for characterization of the performance. Performance of the pattern classification systems is usually measured as the percentage of the samples it can correctly label in an unseen situation. Very high accuracy is expected for many of the pattern recognition applications like character recognition.

Probabilistic techniques are popular for modeling distribution of examples, thereby describing theoretically optimal classifier systems. Major part of pattern recognition algorithms focus on such techniques. Most of these results are applicable only to the two class classification problems. However, most of our pattern recognition problems are multiclass in nature. With the availability of computational power, large class pattern recognition systems are getting more and more attention in recent years. With mounting expectations on the performance of these systems, classifier design is getting addressed from a different angle.

In machine learning literature, binary Bayesian classifiers got shadowed by the practical applicability of Neural Networks and Decision Trees in mid 1980s and early 1990s [19, 60]. The success of these algorithms is due to the computationally efficient numerical algorithms which solved hard optimization problems with acceptable accuracy. Advances in machine learning in the late 1990s argued for generalizable learning schemes, with theoretically

interesting properties [81]. Support Vector Machines (SVM) are shown to over-perform the neural networks in most learning problems.

An important parallel thread of interest to the pattern classification problem is that of identification of the appropriate features, given a set of measurements. Available measurements are either transformed to a new set of features or an important subset is selected for efficient and accurate performance [51, 18].

Computational complexity of the pattern classification algorithms has become critical for large class pattern recognition problems. It is argued, that instead of a direct multiclass classifier, use of multiple small classifiers can provide accurate and efficient pattern classification [71]. These modular classifiers are often hierarchically arranged for building the composite classifier. Hierarchical classification systems are based on the popular *divide-and-conquer* strategy, which breaks down the huge and complex task into small manageable and easily-solvable sub-tasks. The results from these subtasks are later integrated to solve the complete task.

1.1 Hierarchical Learning and Classification

Hierarchical classifiers present the following advantages, because of which they are highly preferred:

- Hierarchal classifiers use an intuitive representation, where simple and broad classifications happen at shallow nodes of the hierarchy, and precise and complex decisions are taken at the deeper nodes of the hierarchy.
- When compared to other ensemble methods, hierarchical classifiers use simpler and lesser number of classifiers.
- Hierarchical classifiers exhibit high classification speeds. Only selected classifiers from the set of all classifiers is evaluated on a given sample, resulting in low classification time.
- Hierarchical methods use lesser space compared to other ensemble methods.
- Hierarchical classifiers allow multi-labeling of classes, which is difficult to realize in direct multiclass approaches.
- Hierarchical classifiers are highly flexible in nature due to their modularity. This allows the usage of different features and classifiers at each node, providing a scope for improving the classification performance of the classifier to a good extent.
- Since the architecture is modular, any other multi-stage architecture can get well with the hierarchical architecture, resulting in hybrid architectures.

- Because of their low space and time requirements, they are the most scalable approaches when compared to any other multi-classifier architectures.

- The precise decision of a class is made at the leaf nodes in the hierarchy, and also this has a higher probability of misclassification compared to the root nodes. This makes it easy to obtain the second/k-th alternate labeling of the sample by traversing up the evaluation path, which is useful if there are corrective algorithms in the post-processing stage (eg. useful for spell checkers).

Building a hierarchical classifier involves two steps (a) designing the hierarchy (b) designing the internal nodes. There are a variety of approaches proposed for both the tasks. All the approaches may be generalized to recursively partitioning the feature space into smaller and smaller subsets and representing each partition as a node in the hierarchy using a learnt classifier. Broadly, the hierarchy building approaches can be divided in the following ways (a) Feature based manual division, (b) Feature based automatic division (c) Class based manual division and (d) Class based automatic division.

Recent trend has been to prefer class-based hierarchies compared to feature based partitioning. The idea of feature-based hierarchies had been of much attention for several decades. However, the performance they exhibit on problems with numerical features was always shadowed by numerical-optimization based approaches like Artificial Neural Networks, Support Vector Machines etc. This is due to the fact that the decision tree design methodologies could not fully realize the benefits of a hierarchical system. The recent trend has been to look into the capabilities of hierarchical systems and use a completely different direction in their design, which is class based partitioning.

Hierarchical classifiers have been applied successfully in various application areas like text categorization, data mining and computer vision to an extent. A summary of the work done is presented in Table 1.1. Section 1.6 discusses the conventional hierarchical approaches, and the problems associated with them. Section 1.4 presents an overview of the recent work done on the hierarchical classifiers using class based partitioning.

1.2 Challenges in Design of Hierarchical Classifiers

As discussed above, design on high-performing hierarchical classifier needs to address two sub problems: (a) Design of the Hierarchy i.e., design of the order in which the problem gets successively solved. (b) Design of the internal nodes i.e., the discriminant capability which the node needs to learn from the examples.

The following are some of the challenges in designing hierarchical classifiers.

- At every stage of the decision making, the selection of the best attribute or best set of classes is difficult without enough look ahead, which is practically impossible.

Core Idea	Literature	Remarks
Manual Partitioning	[58, 12, 72, 79, 59, 15]	Useful only when the hierarchies are already available, which is rarely possible.
Automatic Partitioning	[90, 10, 24, 70, 89, 82, 14, 13, 53, 54, 83, 76, 21, 37, 32]	Partitions are data dependent, and are effective for many applications
Improved Accuracies	[54, 70]	Class based hierarchies perform better than simple multiclass approaches like Naive Bayes.
Theoretical Analysis	[82]	Hierarchical classifiers are shown to be faster and efficient for pattern classification
Base Classifiers other than SVM	[13, 54]	The performance with base classifiers other than SVM is shown to be low
Hierarchy of Features	[43, 35, 84] [30, 8, 6] [69, 36] [8, 63, 62] [80, 33, 78, 87]	These are classical methods, but they suffer from large number of nodes, and overfitting.
Vision Applications	[65, 88, 39, 34, 22, 38]	Mostly two stage classifiers are used, with good success.
Text Classification	[15, 79, 52, 72, 11, 20]	The problem is inherently hierarchical, and tree based classifiers are well suited.
Data Mining	[32, 61, 91]	Usually there is large amounts of data to be mined, and hierarchical approaches proved extremely fast and efficient.

Table 1.1: Summary of literature available on hierarchical classifiers

- Classifier architecture itself need to adapt to meet the data for better performance. Thus the problem of design has many more free parameters.
- Classification function at each node can become quite complex if one looks for a complex decision at each node. This will result in loss of generalization.
- Most of the problems in the class of design is NP-Hard and obtaining optimal solutions efficiently is not easy.
- Since the design of classifiers are relatively new area, techniques have not matured enough.

Though many of these challenges were known for quite some time, they are becoming critical to the design of high performing large class classifier systems. When the data is numerical, decision trees may not result in the best performance. However, the large amount of research in the area of decision trees will help in addressing the problem of designing hierarchical classifier for efficient and accurate pattern classification.

1.3 Decision Trees and Conventional Approaches

Decision trees are hierarchical classifiers built using feature based partitioning. Each node in the tree performs a test on a single, multiple or all attributes of the instance. The tests on an instance start at the root node, traverse the tree through a subset of the nodes depending on the decision at each node, and reach a leaf node where a class is assigned to the instance. Decision trees are most suitable for learning the functions with discrete output where the instances are represented using attribute-value pairs.

Optimal Decision Trees Constructing an optimal decision tree is shown to be an NP-complete problem in [42]. Because of this, many feasible solutions to the decision tree building have been using a greedy approach. The decision tree building algorithms can be classified broadly into *Top-Down, Bottom-Up, Hybrid* and *Grow and Prune*. Top-down approaches were most researched and preferred ways of inducing the decision trees.

Top Down Growing In Top-Down approaches the dataset is split into partitions at each stage of tree building. The rule used to split is what decides the structure and performance of the final classification tree. After splitting, certain nodes are labeled as terminal nodes, and are assigned a label of a class. A perfect split is a split where all samples belonging to one class go to the left and the other go to the right. However, in practice, this is not possible, and the most probable class reaching a particular node is assigned to the node, if it is chosen as a terminal. Most of the research in top-down induction of decision trees has concentrated on using the splitting rules. Also, the splitting rules can be univariate or multivariate.

- Information theory based tree induction: mutual information by [43], information gain by [35, 84] and gain ratio
- Distance based tree induction: Gini Index by [30], Twoing Rule by [8], Margin based Twoing-rule by [6]
- Performance Based tree induction: Minimum Description Length by [69], Inaccuracy, sum-minority, max-minority by [36]
- Other popular approaches: linear discriminants, hill-climbing by [8], OC1 by [63, 62],
- Hybrid approaches: Perceptron Trees by [80], Neural Trees by [33, 78], Omnivariate decision trees

Stopping the Growth An important aspect in top-down induction of decision trees is to know when to stop growing the decision trees. Certain constraints on the tree growth are applied so that the tree size is limited, which is expected to improve the generalization. Restrictions on the number of samples reaching a node, thresholds on the impurity measure have been used to limit the tree growth. Pruning of the tree is also used after the tree is completely grown to obtain the trees of right size.

Comments When compared to the modern classification approaches like Neural Networks, Boosting, Multiclass-SVM etc, they are found to be inferior in the following respects:

- Decision trees do not have a restriction on the leaf-node labeling, and hence the number of leaf-nodes usually are much higher than the number of categories.
- Decision trees are well suited for problems with categorical features, however are not really popular in approaches with real-valued features. In real-valued problems, classifiers like neural networks and SVMs have shown much superior performance.
- Decision trees usually try to solve the classification task using simple rules or linear classifiers, which results in large number of nodes even in slightly complex classification tasks.
- Decision trees are observed to overfit as they try to learn the training data completely and accurately.
- Class-based division of nodes is more intuitive than the Feature-based division.
- Class-based division results in a constant classification complexity, where as the feature-based division results in different hierarchies, with different sizes at each time, all of which are much larger than the class-based divisions.

- The class is not known until a leaf node is reached, where as the group of classes to which the sample might belong to, is always available in the case of class-based hierarchies.

- The problem of obtaining an optimal-decision tree is NP-hard, and hence only approximate solutions exist to resulting in sub-optimal trees [42].

1.4 Hierarchy Design Methodology

In [54], a hierarchical classification method based on an earlier formulation of Generalized Associative Modular Learning Systems (GAMLS) [53] is presented. In this, the hierarchy is generated automatically from the dataset, resulting in a set of simple binary problems, each with its own feature space. The resulting hierarchy has $N-1$ nodes for an N class problem. They use a Bayesian classifier as the binary classifier at each node, and the N leaf nodes in the tree determine the label of the sample. The data is first partitioned recursively at a node n into sets of classes Ω_{2n} and Ω_{2n+1}, until only one class is left in each partition. For a given set of classes Ω, the algorithm partitions the set into two subsets Ω_α and Ω_β. They model the base classes $\omega_i \in \Omega, i = 1 \cdots N$, as normal distributions with mean μ_i, covariance Σ_i and prior P_i. The meta-classes are modeled as normal distributions with mean μ_γ, Σ_γ and prior P_γ, where $\gamma \in \{\alpha, \beta\}$. Using the within class scatter, $W = P(\omega_\alpha \Sigma_\alpha + P(\omega_\beta \Sigma_\beta$ and between class scatter $(\mu_{alpha} - \mu_{beta})(\mu_{alpha} - \mu_{beta})'$, the fisher projection is defined as $J(w) = \frac{w'Bw}{w'Ww}$. The partition is found between the subsets α and β, such that the fisher linear discriminant is maximized. This can not be done directly as the labeling of classes is not available. Therefore, an iterative algorithm based on deterministic annealing is used where the class parameters are updated using the fisher projection, and the fisher projection for next iteration is computed. This is done until no significant increase in the discriminant is found, and the entropy (used as the temperature term) is below a certain threshold.

Ghosh *et. al* [13] proposes hierarchical space decomposition approach based on max cut for automatically designing the hierarchy for a classification task. Support Vector Machine (SVM) classifier is used as the base classifier in all the nodes of the hierarchy. The objective of this decomposition is to avoid the problem of complex decompositions obtained by other methods like ECOC, one-vs-rest and pairwise classifiers. SVM and max-cut match well to each other. Max-cut is formulated to obtain the maximum separation between the clusters, and SVM is trained to obtain a large margin in this case. The amount of training time and the difficulty of the optimization are less because the well-separated output of max-cut aids the quicker training of SVM. In a max-cut problem, an undirected graph is partitioned into two groups. The cut between the groups has the maximum weight. The graph $G(N, E)$ is modeled with classes as the set of nodes N and edge weights w_{ij} as the average Kullbeck-Liebler distance between the classes ω_i and ω_j. The weight of the cut K in this graph $W(K) = \frac{1}{2} \sum_{i<j} w_{ij}(1 - x_i x_j)$. The resulting quadratic programming

problem is to maximize $W(K)$ by assigning the values of $\{+1, -1\}$ to the variables x_i and x_j. This problem is an NP-hard problem, whose solution is approximated by converting this quadratic integer programming problem into a semi-definite programming problem and is solved using an extended interior point algorithm. The results showed that their previous approach using BH-SVM obtained better results than the Max-cut formulation, however max-cut training times were much faster, and resulted in lesser number of support vectors.

Another method [82] uses a clustering based approach called DB2 (Divide By 2) to design the hierarchy. The hierarchy is a binary tree with SVMs as the base classifiers at each node. Divide by 2 essentially uses clustering approaches to divide the dataset into two partitions at each stage. Three clustering methods were discussed. One is the standard K-means, used to cluster the dataset into two clusters (K=2). The second one is called spherical shells. In this, the mean M of the complete data is computed, and a sphere of radius M and centered at origin is used to separate the dataset into interior and exterior clusters. The classes with means falling inside the sphere form the interior cluster, and with means falling outside form the exterior cluster. The third approach is balanced subsets approach, where the classes are divided such that the number of samples in both the partitions is almost equal. This method was meant to obtain shorter and well balanced trees for faster classification. The DB2 algorithm may be summarized as, partition the data, train the SVM and repartition the partitions recursively until only one class is left in both the partitions. DB2 is faster than all multiple classifier approaches. The training times of DB2 algorithm were computed in terms of the complexity of training an individual SVM suing the SMO algorithm, and is shown to be same as training all the pairwise classifiers in the usual case. A generalization analysis of the DB2-SVM is presented. As the generalization bound depends on the depth of the tree, DB2-SVM is bound to perform better. However, the margins are unknown and any concrete comparison of DDAG and DB2-SVM is not possible. An adaptive approach was mentioned, where a separate kernel is selected for each binary classifier. Experiments reveal that DDAG performs better, however the testing times are shown to be less compared to that of DDAG.

A Self Organizing Map (SOM) for partitioning the data into two at each level of hierarchy building is reported in [14]. Two types of SOM are used, a normal SOM and a kernel SOM (K-SOM). The dataset is first presented to a SOM algorithm and the two dimensional mapping is obtained. This map is then used to partition the dataset, in two ways. A man-based approach, where a human studies the output of the SOM and divides the map into two partitions. This division is mapped back on to the input space and a SVM is trained. The other method is to automatically cluster the SOM output such that the distance between means of the clusters is maximized and the variance within each class is minimized. They present the complete derivation of K-SOM. The accuracies obtained using the algorithm are compared with other approaches like Neural Networks, 1-v-1 SVM, 1-v-All SVM. Of all these approaches, K-SOM based human grouping is observed to show best accuracies, and K-SOM-SVM in general was shown to perform better than all other approaches on the

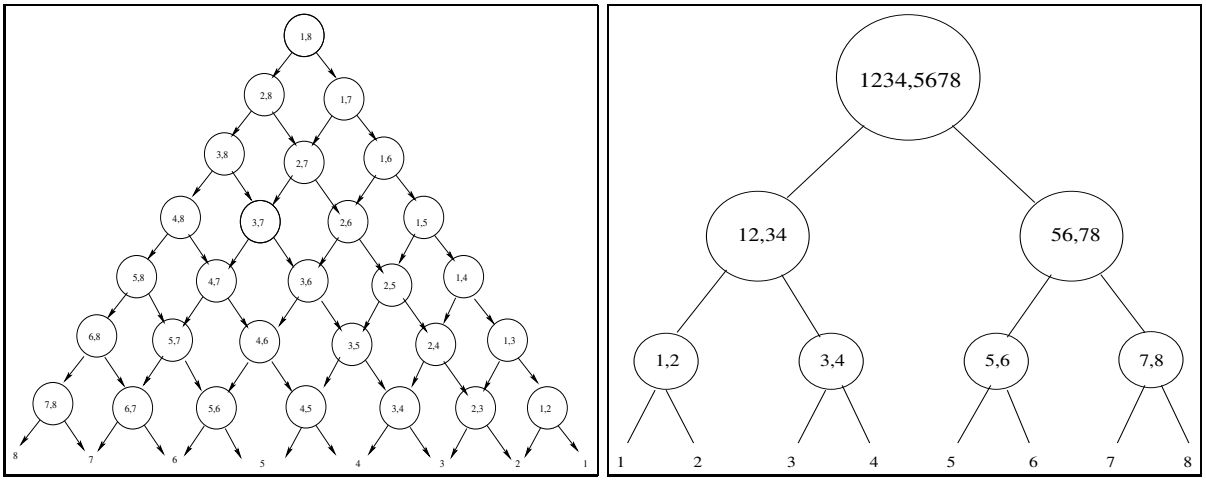

Figure 1.1: The Decision Directed Acyclic Graph and the Binary Hierarchical Classifiers

selected datasets.

A taxonomy based approach for building hierarchical classifiers for known hierarchies, which builds on direct multiclass approaches of Crammer and Singer is reported in [10]. They suggest that direct approaches do not consider the relation between different categories in text classification (or in any such problem), and hence hierarchical approaches are bound to perform better. The multiclass algorithm of Crammer and Singer is formulated as a quadratic convex optimization problem, similar to the two class SVM. The one-vs-rest approach is inherent in the definition of their classification function. The problem is formulated as a combination of one-vs-rest classifiers, and the classifier with largest confidence on the sample (distance from the margin) is used as the label for the sample. Cai [10] generalizes this approach where the formulation is no-more one-vs-rest, but based on a code-vector based approach. For example, if a hierarchy has 10 classifiers, each class has a code vector of length 10. Of the 10 classifiers, if the classifier is on the evaluation path of the class ω_i, then it is represented as 1 in the code vector of ω_i. This is posed as a dual quadratic problem, as in the case of binary SVMs and is solved using an iterative optimization algorithm discussed in the paper. The accuracies obtained using this algorithm are measured in terms of accuracy, precision, taxonomy-based loss and parent accuracy. The hierarchical SVM is shown to be significantly better than the flat approaches.

1.5 Decision Directed Acyclic Graph and Binary Hierarchical Classifier

There are two important classes of hierarchical multi-classifier systems, that of special interest to the work reported in this thesis. They are Decision Directed Acyclic Graph (DDAG) and Binary Hierarchical Classifier (BHC). A typical configuration of these classifiers to solve an 8-class problem is shown in Figure 1.1(a) and Figure 1.1(b). A node with a, b in these structures represent a binary classifier trained to solve the problem of separation of classes ω_a and ω_b.

DDAG Given a test sample for classification, it is first presented to the root node of the DDAG for the decision. The decision made at each node of the classifiers is not which class the sample belongs to, but which class the sample *does not* belong to. One of the classes is rejected as a feasible class at the root node and the sample thus reaches the root node of the left or right sub-DAG. Another class rejection takes place at this root and the sample follows an edge leading to one of the sub-DAGs of the current root node. Thus the sample visits $N-1$ nodes before it reaches the leaf of the DAG. Testing complexity of the DDAG is $O(N)$ while the number of nodes or classifier is $O(N^2)$. The sample is assigned the label corresponding to the leaf node it reaches. A DDAG architecture, with nodes as pairwise classifiers using SVM was proposed in [67]. The DDAG classifiers are shown to outperform almost all multiclass classifier systems [41]. However, there has been practically no attempt towards the design of the DDAG classifier for improving the performance.

BHC A BHC arranges $N-1$ binary classifiers in the form of a binary tree, such that there are N leaves where the sample gets its labels assigned. Usually in all the BHC algorithms, the dataset is recursively partitioned into two subsets recursively, and a classifier is learnt [54] to implement the partitioning. Different clustering algorithms have been employed to partition the data into two clusters. A BHC has $O(N)$ nodes and can evaluate the label of a test sample in $O(log(N))$ time on an average. Different approaches have been tried out in building the hierarchy of mutually exclusive subsets of classes at every stage of the algorithm.

1.6 Selected Applications of Hierarchical Classifiers

Main application areas where hierarchical classifiers have been popular are text classification, computer vision and data mining. In each field, hierarchical classifiers have been employed to achieve various ends, like improvement in computational speed to emulating a multiclass classifier.

1.6.1 Computer Vision

Face detection is a two class problem, however, although the set of positive samples is clearly defined, the set of negative samples is highly varied, and can not be modeled easily. Hierarchical classifiers have been used to solve this problem and various successful methods are discussed here. Component based models are observed to produce better results when compared to global detection and recognition approaches. This is same as the case in large class recognition problems, where it is difficult to model the classification as a single task.

[38] presents the support vector learning in a format useful for hierarchy design algorithms. The task of face detection under cluttered backgrounds is solved using a hierarchical approach, which uses SVMs as the base classifiers. The margin M obtained in a support vector machine can be expressed as $M = \frac{1}{\sqrt{\sum_i \alpha_i}}$, where α_i are the coefficients of samples obtained by support vector training. The margin indicates the separability of the data. Also, the expected error probability of a support vector can be measured as $EP_{err} \leq \frac{1}{l} E\left[\frac{D^2}{M^2}\right]$, where D is the smallest sphere containing all points in the dataset. Components from an image are extracted automatically such that they maximize this measure. An initial seed region is selected, which is grown in various directions, computing the margin bound each time. Many linear SVMs are trained to identify the individual components, for eg. expert in eyes, nose, etc. These detected regions are then sent to a combining classifier for resulting in the complete detection of a face. This system was observed to perform better than the single stage complete face detection approaches.

[65] discusses a hierarchical SVM based approach for face membership authentication. They use an Locally Linear Embedding(LLE) based approach to build the hierarchy by clustering it into two parts at each stage. LLE recovers global non-linear structure from the locally linear fits. The hierarchy expected to be obtained in this case is different from the previous discussion, as it has only positive or negative labels, with hundreds of leaf nodes. Each node in the tree is a LLE based representation of the local neighborhood with members and non-members and an SVM is used on that.

[88] uses a hybrid hierarchical classifier with a combination of linear and non-linear SVMs. This was used to obtain speed in calculation. In face detection, there is a very small region in the image with face, where as most of the image is background. The linear SVMs were used to quickly reject the background, and only when a sample gets classified positively with all the linear SVMs, a very complex Non-linear SVM is used to confirm the decision of the image patch being a face.

1.6.2 Text Classification

Text classification is an area where natural hierarchies of classes exist. It is typically the case that documents are hierarchically organized under various topics. In these cases, the building of hierarchies is driven by the human labeling of the documents, and hence is more

accurate. The concentration would be on improving the node level classifiers using better classifiers, or generalizing the knowledge onto other unlabeled samples [46].

However, when it comes to certain document collections where such man made hierarchies are not available, automatic building of the hierarchies is necessary. A good example of this is the classification of world wide web [20, 28]. Most of the literature in text classification discusses the better categorization obtained due to hierarchical classifiers, both from the suitability to the problem and performance perspectives.

[52] discusses the preferable of the hierarchical methods for text classification over many existing flat approaches. Flat approaches are not suitable as the number of classes in text categorization are usually large, and also the feature definitions are not really suitable for a simple flat classifier.

1.6.3 Data Mining

Many real-world data mining applications involve millions or billions of data records where even multiple scans of the entire data are too expensive to perform. Hierarchical methods are useful in reducing the complexity of training and testing time for classifiers.

In [91] a new method called Clustering-Based SVM (CB-SVM) is presented. This method is designed for handling very large data sets. CB-SVM applies a hierarchical micro-clustering algorithm that scans the entire data set only once to provide an SVM with high quality samples that carry the statistical summaries of the data such that the summaries maximize the benefit of learning the SVM. CB-SVM tries to generate the best SVM boundary for very large data sets given limited amount of resources. Their experiments on synthetic and real data sets show that CB-SVM is highly scalable for very large data sets while also generating high classification accuracy.

A new technique for multi-way classification which exploits the accuracy of SVMs and the speed of Naive Bayes classifiers is proposed in [32]. The Naive Bayes classifier is used to compute a basic confusion matrix, which is later used to reduce the number and complexity of the two-class SVMs used in the second stage. During testing, the prediction of the Naive Bayes classifier is obtained, and only a subset of the two-class SVMs is applied selectively.

1.7 Design of Class Hierarchies for Efficient Classification

Many practical pattern recognition applications such as character recognition, biometrics etc, have multiple classes of patterns to recognize. Binary classification is a well researched area with exhaustive theoretical analysis and extremely useful results [19]. Many of these results can be extended for multi-class recognition problems. Few direct extensions of binary classifiers exist for multiclass classification. However, it is usually preferred to solve the multiclass classification using a combination of binary classifiers. Such combinations are found to outperform the direct multiclassifiers empirically [41] and theoretically [67].

Multiple classifier combination schemes can broadly be divided into two classes – two stage and hierarchical. Both of these approaches split the problem into multiple simple (often binary) problems and integrate their results. In two stage scheme, a given sample is presented to all the classifiers, and a majority vote is used to assign a label. Hierarchical systems for classification can be modeled as a tree or a graph. Classification is done by following a path from the root node to the leaf node of the classifier. In general, hierarchical systems have smaller number of classifiers. Hierarchical classifiers can often classify the samples faster than the two-stage approaches.

Recent trend has been to prefer class-based hierarchies compared to feature based partitioning for the design of hierarchical classifier systems. Feature-based hierarchies are popular for several decades. However, the performance they exhibit on problems with numerical features was always dominated by numerical-optimization based approaches like artificial neural networks, support vector machines etc. The reason for this was that the decision tree design methodologies could not completely exploit the benefits of a hierarchical systems.

This work focuses on building hierarchy of classes, rather than features, for designing efficient and accurate classifiers. Deviating from the decision tree paradigm, we depend on a class-based hierarchy for the classifier design. Building class-based hierarchy is superior to that of feature-based hierarchy because of the following reasons:

- Class-based division is more intuitive than feature based division as natural groups of classes exists for many problems. It can be observed evidently in applications like text categorization and character recognition.
- Class-based division results in a predictable number of component classifiers unlike feature-based hierarchies.
- When large number of classes exists when compared to features (eg. OCR problems), feature based partitions are insufficient to express the complete classifier.
- Class-based partitioning schemes employ the natural labeling of the samples for grouping. While feature-based grouping needs not result in directly useful partitioning at every stage.

Graphs and Trees are effective means for integrating results of multiple classifiers. In these structures, nodes are classifiers and directed edges provide the path along which evaluations are made, until a terminating node is reached. A classical example of such a classifier is a decision tree. Optimal design of a decision tree is an NP-Hard problem, and greedy algorithms are popular for its design, even though they are sub-optimal. A Decision Directed Acyclic Graph (DDAG) is a generalization of decision tree. It can be effectively be used for building a high performing classifier system [67]. Classification with tree-based classifiers is faster but often less accurate compared to the DDAG classifiers. Since any tree classifier can be considered equivalent to a binary classifier, we limit our attention to the Binary Hierarchical Classifiers(BHC).

1.8 Problem Statement

Conventional pattern recognition systems were designed by solving two popular sub-problems.

- Finding an appropriate decision boundary for a set of classes and a numerical training algorithm to obtain these boundaries.
- Finding a set of features (a subset of the existing ones or a linear or nonlinear combination of the measurements) which can solve the problem in a better manner.

Deviating from these well explored paradigms, this thesis addresses issues associated with the design and ordering of building blocks needed for the hierarchical classifiers. The design and analysis procedures proposed in this thesis are complementary to the results available in conventional pattern recognition literature. For example, the individual nodes of the hierarchical classifier can be any of the popular pattern classifier with its favorite training and evaluation scheme. In general, for the discussions in this thesis, we assume that each of these nodes are Support Vector Machines (SVM), which are shown to be one of the best for classification and generalization [19].

This thesis addresses the design problem of two important hierarchical classifiers – DDAG and BHC. To be more specific, the following problems are addressed.

Design of DDAG Given a collection of training samples from multiple classes, design an optimal DDAG to maximize the classification accuracy of the whole classifier system.

This has three individual sub-problems.

- Maximization of the performance of the individual nodes such that the combined system provides better accuracy.
- Formulation of the problem to find the optimal class sequences (relative positioning of the nodes in the graph) for the maximal performance of the DDAG classifier.
- Identification of appropriate numerical algorithms for design of the DDAG classifier to improve the performance of the classification system.

Design of BHC For a given set of classes, BHC provides one of the best training, test and storage complexities. However, their performance is found to be inferior to the DDAG. Given a set of training examples, we are interested in obtaining better accuracy for the BHC without considerable loss in its advantages relating to storage and evaluation time.

In this thesis we attempt to

- Formulate the class-based partitioning problem as a graph-based hierarchical cut detection problem and show that in a general situation (independent of the data) no clustering algorithm can be superior to others.

- Propose a new partitioning scheme which does not insist on mutually exclusive partition for the design of the BHC.
- Propose a new design algorithm which combines the higher performance of the DDAG with the low storage complexity of the BHC to build configurable hierarchical classifiers.

1.9 Major Contributions

Hierarchical classifiers are usually built of fewer number of classifiers. Only a subset of these classifiers evaluate a sample, resulting in low classification time. The topology based on which the classifiers are combined is generally referred to as the architecture of the combined classifier. Most hierarchical algorithms follow a binary tree architecture. Algorithms for design of hierarchical classifiers focus on evolving the tree by learning the classifiers at each node. This is typically achieved by using classifiers which give better generalization at each node. Popularly, classifiers like Support Vector Machines (SVMs) are preferred at the individual nodes. Existing hierarchical approaches often overlook the possibility of selecting an appropriate architecture. This thesis describes the design, analysis and application of the hierarchical approaches for solving multiple-class classification problems. Emphasis is on the analysis and design of the classifier system by defining an appropriate class hierarchy.

We argue that hierarchical classifiers are highly suited for many of the pattern classification problems. For applications like character recognition (specially in Indian Languages), with large number of classes and high expectation of classification performance, DDAG classifiers are found to be highly suitable [9]. We demonstrate the application of the DDAG classifier on character recognition applications. A direct use of the DDAG architecture and trained nodes may not result in the best design of the DDAG classifier.

The performance of an ensemble of pairwise classifiers to emulate multiclass classification system depends on parameters from data, individual pairwise classifiers, and the architecture used. We argue that by appropriately rearranging the nodes in the DDAG classifier, one can obtain a better classifier. When the design is based on only the *apriori* probability of the classes, we show that [57] an optimal design maximizing the classification accuracy of the DDAG can be obtained using a greedy algorithm. In a general situation, the optimal design of the DDAG is shown to be NP-Hard and practically impossible to compute for large class problems. We propose approximate greedy algorithms for the design of the DDAG classifier system. A novel formulation for the design of multiclass classifier is presented. Algorithm for designing optimal multiclass classifiers using the DDAG architecture is proposed.

A critical step in the design of the Binary hierarchical classification, is the choice of the partitions at the successive nodes in the hierarchy. We argue that none of the partitioning schemes can work well on all data sets because of the lack of any valid assumption on the

distribution of the classes in the feature space (Note that there could be valid structure within the class, even when there is no structure for the distribution of the classes). We propose a new partitioning scheme which does not insist on mutually exclusive partition for the design of the BHC. We also show that the complexity of the test still remains $O(\log N)$ in average case. The scheme looks at the generalization performance of various possible partitioning schemes and their improvements by non-committed on special selected classes.

The performance of the hierarchical classifiers depends on performance of the individual classifiers and the overall combination architecture. We propose a scheme that incorporates both these factors in designing a hierarchical architecture. The proposed scheme allows selection of a suitable architecture at each node of the tree [55]. A mechanism to specify the suitability of an architecture, as per the requirements like high-accuracy or low-classification time, is provided by combining the advantages of two algorithms BHC and DDAG. As an instantiation of this method, an algorithm is proposed for evolving a configurable tree based architecture. The effect of the configuration parameter on the design is studied. It is shown that use of hierarchical combination of complex classifiers, suits the problem of large class character recognition [56]. This results in classifiers with better generalization and with less space and time complexities.

1.10 Organization of the Thesis

This chapter describes the setting of the thesis with the general background material needed for the thesis. Advantages of the hierarchical classifiers is highlighted for pattern classification applications. Feature-based and class-based hierarchies are described and an overview of the feature-based hierarchy for decision trees is described. Two hierarchical classifiers (i.e., DDAG and BHC) of special interest are introduced in this chapter. Motivation behind the present work, the problem description and the major contributions are described. Chapter 2 provides a basic introduction to the background information required for the rest of the thesis.

Chapter 3 describes the node-level problem of a DDAG classifier. DDAG classifier is composed of large number of pairwise classifiers. Design of the individual nodes is the focus of this chapter. A feature selection schemes is employed for improving the performance for the individual nodes. Performance of DDAG classifiers are shown on standard data sets with special emphasis on the applications in Indian Language OCRs. A further dimensionality reduction is proposed to reduce the storage complexity of the classifier.

Chapter 4 addresses the problem of design of DDAGs by rearranging the relative positions of the classifier. This problem is posed as an optimization problem of misclassification probabilities. In a restricted setting, it is shown that the optimal design can be obtained with a polynomial time algorithm. However, in a general setting, this problem is NP-Hard. Conventional DDAGs are capable of providing highly accurate classification with limited evaluation complexity. However, they are unable to provide a 'reject-class' for improving the

performance. An improved DDAG scheme is proposed in this chapter. Approximate algorithms are proposed for this. It is experimentally shown that these algorithms consistently provide designs which are close to the average case.

Chapter 5 introduces an algorithm for partitioning a set of classes into multiple subsets. Because of the lack of any systematic structure, it is shown that none of the partitioning algorithms could really result in partitions with very high margin for any general dataset. A new scheme, which divides the classes into two possibly overlapping subsets is proposed as an attempt to maximize the margin and thereby the performance.

Chapter 6 discussed an approach which exploits the advantages of both BHC and DDAG classifier. The DDAG classifier has higher accuracy with high storage complexity, while BHC is extremely efficient in with reasonably high classification accuracy. A configurable hybrid classifier is proposed for multiclass classification problem. This classifier employs SVMs at most of the nodes and DDAGs for implementing a single multiclassifier for a subset of classes.

Chapter 7 concludes with the summary of the thesis and scope for future work.

Chapter 2

Preliminaries

2.1 Pattern Classification

Pattern Classification is "the act of taking in raw data and making an action based on the category of the pattern" [19]. The process of pattern classification involves three main steps: preprocessing, feature extraction and classification. The preprocessing step involves removal of irrelevant data from the raw data for simplifying the subsequent operations. The feature extraction step is to extract relevant properties from the raw data suitable for classification. The classification step uses these features to categorize the raw data into one of the known categories. These categories are referred as classes, and the raw data as samples.

The Problem of Pattern Classification Formally, the task of pattern classification can be defined as assigning a label to a given sample, represented numerically using a vector x, using one of the possible labels from the set of all classes $\Omega = \{\omega_1, \omega_2, \cdots, \omega_N\}$

2.1.1 Posterior Probabilities and Bayesian Error

The categorization of the sample x into one of the classes $\omega_1, \omega_2, \ldots, \omega_N$ is usually done by calculating the probability that the sample belongs to a class ω_i. This is called the posterior probability, and is represented using $P(\omega_i|x)$. The posterior probability that x belongs to all the classes is computed, and the sample is assigned a label of the class ω_i whose posterior probability $P(\omega_j|x)$ is maximum. The posterior probabilities can be calculated using the "Bayes rule" as follows:

$$P(\omega_i|x) = \frac{p(x|\omega_i)P(\omega_i)}{p(x)}$$

where, $p(x|\omega_i)$ is called the likelihood probability, which is the likelihood of a sample x occurring from a class ω_i. $P(\omega_i)$ is called the apriori probability of the occurrence of a

sample from the class ω_i. Bayes rule can also be expressed as:

$$posterior = \frac{likelihood\ X\ prior}{evidence}$$

Bayesian error: "Error" is said to have occurred when the sample is categorized into a class which it does not belong to. The maximum aposteriori method described above, when used for classification has the minimum error. This is called the "Bayesian error". For instance, let us consider the categorization of the sample x into one of the classes ω_1, ω_2.

$$\begin{aligned} P(error|x) &= P(\omega_1|x)\ if\ we\ decide\ \omega_2 \\ &= P(\omega_2|x)\ if\ we\ decide\ \omega_1 \end{aligned}$$

$$P(error) = \int_{-\infty}^{\infty} P(error|x)p(x)\ dx$$

$P(error)$ gives the probability of error. This is minimized by using the maximum aposteriori method because $P(error|x) = \min[P(w_1|x), P(w_2|x)]$. Hence the Bayesian decision is the optimal decision with minimum error called the Bayesian error.

2.1.2 Learning and Generalization

A classifier learns to predict the class ω_i to which the sample belongs to, during the training phase. In the training phase, a large number of examples, with known labels are presented to the classifier, and an algorithm gains classification knowledge, and stores it as a set of internal parameters. The error that a classifier makes while training is called the empirical error of the classifier.

The capacity of classifier to correctly predict the label of a sample, unseen during the training phase, is defined as generalization. Ideally, a classifier should have a low training error and high generalization for best performance.

Few definitions of frequently used terms in pattern recognition problem are given below:

Dataset: A dataset is a collection of labeled samples. There are three kinds of datasets – (a) training data (b) testing data and (c) validation data. The training data is used for learning the parameters of the classifier, and the testing data is used to evaluate the performance of the classifier. The validation data is used in some cases to select a good set of parameters, when multiple sets parameters are possible. The learning and generalization capability of a classifier highly depends on the training data used for each class.

Training: Training is the process where the classification parameters of a classifier are estimated by observing a set of labeled examples. The trained classifier can be used to predict the label of the samples unseen while training. Usually, the classifiers cannot

completely learn the input samples, and there is an error associated with the training process. This is called the training error.

Testing: Samples unseen during the training phase are presented to the classifier to obtain their label. The label predicted by the classifier may not always be correct because of limitations of the training algorithms and data. A classifier can be evaluated based on the probability of being correct on a set of unseen samples. The error that a classifier makes on unseen samples is called the testing error.

Overfitting: Usually, classifiers are preferred with low training error as well as low testing error. If the training error of the classifier is reduced by training excessively, the classifier tends to "memorize" the training samples and fails to perform well on unseen samples. This is called overfitting. This is an important characteristic of a training phase which must be avoided.

2.1.3 Performance Measures

Computation of the Bayesian error of a classifier makes little sense in the practical scenario, as little is known about the probability distributions of the classes. However, to pick a good one out of many different classifiers, a set of performance evaluation measures are used. Among the available performance measures, three popular measures are described below [19].

1. **Accuracy:** This is defined as the ratio of the total number of testing examples classified correctly to the total number of test examples.
2. **Precision:** This is the ratio of positive examples classified correctly to the examples classified positive, when tested on unseen samples.
3. **Recall:** This is the ratio of positive examples classified correctly to the number of known positive, when tested on unseen samples.

2.2 Support Vector Machines(SVMs)

Support Vector Machines(SVMs) are binary classifiers that were introduced by Vladimir Vapnik [81]. These are methods for learning classifiers from a set of labeled training data. For classification, SVMs operate by finding a hyper-surface in the space of possible inputs. This hyper-surface will attempt to split the positive examples from the negative examples. The split will be chosen to have the largest distance from the hyper-surface to the nearest of the positive and negative examples. Intuitively, this makes the classification correct for testing data that is near, but not identical to the training data.

2.2.1 Objectives

Support Vector machines(SVMs) use the positive and negative samples for a class to find the support vectors, the samples that have high probability of getting misclassified. These support vectors are used to define a decision boundary between the classes such that the margin between the two classes is maximized. Hence it is called a maximum margin classifier.

Consider the following two-class classification problem. Given a training dataset of l independently and identically distributed (i.i.d) samples

$$(x_i, y_i), i = 1, 2...l, y_i \in (-1, 1), x_i \in R^d \tag{2.1}$$

where d is the dimensionality of the dataset. SVM constructs the decision function by finding the hyperplane that has the maximum margin of separation from the closest data points in each class. Such a hyperplane is called an optimal hyperplane. Training an SVM requires solving the following quadratic optimization problem:

Maximize:

$$\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \tag{2.2}$$

subject to the constraints

$$\alpha_i \geq 0, i = 1, 2, ...l \text{ and } \sum_{i=1}^{l} \alpha_i y_i = 0 \tag{2.3}$$

where α_i are the Lagrangian multipliers corresponding to each of the training data points x_i. The decision function is given by :

$$f(x) = sgn(\sum_{i=1}^{l} \alpha_i y_i K(x_i, x)) \tag{2.4}$$

The function K in the equations 2.2 and 2.4, is called the kernel function. It is defined as $K(x, y) = \phi(x).\phi(y)$, where $\phi : R^d \longrightarrow H$ maps the data points to a high dimensional (possibly infinite dimensional) space H. For a linear SVM, $K(x, y) = x.y$. We do not need to know the values of the images of the data points in H.

2.2.2 Relationship to Structural Risk Minimization (SRM)

Consider a machine whose task is to learn the mapping $x \rightarrow y$. The machine is defined by a set of possible mappings $x \rightarrow f(x, \alpha)$, where the function $f(x, \alpha)$ are themselves labeled by adjustable parameters α. The machine is assumed to be deterministic: for a given input x and a choice of α it will always gives the same output $f(x, \alpha)$. The particular value of α generates what we call a trained machine. The expectation of the test error for a trained machine is therefore:

$$R(\alpha) = \int \frac{1}{2}|y - f(x,\alpha)|dP(x,y) \tag{2.5}$$

However, we do not have an estimate of $P(x,y)$. Hence the empirical risk is calculated using

$$\begin{aligned} R_{emp}(\alpha) &= \frac{1}{2l}\sum_{i=1}^{l}|y_i - f(x_i,\alpha)| \\ R(\alpha) &\leq R_{emp}(\alpha) + \sqrt{\frac{h(log(2l/h)+1) - log(\eta/4)}{l}} \end{aligned} \tag{2.6}$$

where h is the non-negative integer called the Vapnik Chervonenkis (VC) dimension. The VC dimension for a family of functions is defined as the maximum number of points that can be shattered by them.

The bound thus calculated on the generalization error is independent of $P(x,y)$. If we know h we can compute the bound over the generalization error of a classifier. The support vector machines choose a function $f(x,\alpha)$ with low VC-dimension h such that the right hand side of the equation is minimized which in turn puts an upper bound over the generalization error by structural risk minimization.

2.2.3 Applications and popularity of SVMs

Support Vector Machines have been successfully applied to a number of problems like isolated hand-written character recognition, object recognition, and face detection to name a few. SVMs are defined for two class classification problems, and are later extended to multiclass classification problems using various approaches. Support Vector Machines have been found to be superior for classification over many traditional approaches like neural networks and KNN.

2.3 Graph Based Classifiers

2.3.1 Trees and Graphs

A tree is a set of nodes and edges with a property that each node has exactly one parent. There is a single node without any parents, called the root of the tree. The nodes without any children are called the leaf nodes. Trees are used for representing classification knowledge, and are called decision trees [8]. Decision trees are hierarchical classifiers built using feature based partitioning. Each node in the tree performs a test on a single, multiple or all attributes of the instance. The tests on an instance start at the root node, traverse the tree through a subset of the nodes depending on the decision at each node, and reach a leaf node where a class is assigned to the instance. Decision trees are most suitable for learning

the functions with discrete output where the instances are represented using attribute-value pairs.

In Top-Down approaches the dataset is split into partitions at each stage of tree building. The rule used to split is what decides the structure and performance of the final classification tree. After splitting, certain nodes are labeled as terminal nodes, and are assigned a label of a class. A perfect split is a split where all samples belonging to one class go to the left and the other go to the right. However, in practice, this is not possible, and the most probable class reaching a particular node is assigned to the node, if it is chosen as a terminal. Most of the research in top-down induction of decision trees has concentrated on using the splitting rules. Also, the splitting rules can be univariate or multivariate.

Graphs is a collection of nodes and edges with each edge being characterize by the nodes it connects. The edges have a weight associated with the homogeneity property between the two nodes it connects. The graphs have been used in pattern classification for clustering the data. These are the unsupervised classifiers. Weighted graphs $G(V, E)$ are represented by a set of nodes V and edges E connecting the nodes in the set V. These edges have a weight associated with each of them.

2.3.2 Graph Partitioning and pattern clustering

In graph partitioning well defined problems are studied which closely resemble the problem of finding good clusterings, in the settings of both simple and weighted graphs respectively. Restricting attention first to the bi-partitioning of a graph, let (S, S^c) denote a partition of the vertex set V of a graph (V,E) and let $\sum(S, S^c)$ be the total sum of weights of edges between S and S^c, which is the cost associated with (S, S^c). The maximum cut problem is to find a partition (S, S^c) which maximizes $\sum(S, S^c)$, and the minimum quotient cut problem is to find a partition which minimizes $\sum(S, SC)/min(|S|; |S^c|)$. The recognition versions of these two problems (i.e. given a number L, is there a partition with a cost not exceeding L) are NP-complete, the optimization problems themselves are NP-hard , which means that there is essentially only one way known to prove that a given solution is optimal, list all other solutions with their associated cost or yield. In the standard graph partitioning problem, the sizes of the partition elements are prescribed, and one is asked to minimize the total weight of the edges connecting nodes in distinct subsets in the partition. Thus sizes $m_i > 0$, i $1, \ldots k$, satisfying $k < n$ and $\sum m_i = n$ are specified, where n is the size of V, and a partition of V in subsets S_i of size m_i is asked which minimizes the given cost function. The fact that this problem is NP-hard (even for simple graphs and $k = 2$) implies that it is inadvisable to seek exact (best) solutions. Instead, the general policy is to devise good heuristics and approximation algorithms, and to find bounds on the optimal solution. The role of a cost/yield function is then mainly useful for comparison of different strategies with respect to common benchmarks. The clustering problem in the setting of simple graphs and weighted structured graphs is a kind of mixture of the standard graph partitioning problem

with the minimum quotient cut problem, where the fact that the partition may freely vary is an enormously complicating factor.

2.4 Feature Selection

Feature selection is an important step of the pattern classification problem. On high dimensionality of the features, the process of pattern classification becomes very cumbersome. Hence there is a need to reduce the dimensions of the features without loss of much of information. Dimensionality reduction techniques such as principal component analysis and Linear Discriminant Analysis transform the features into a lower dimensional space without much loss of information. However, there is another method called the subset selection such as forward search, backward search and Tabu search that can be used to select only a few features that can be helpful in classification.

2.4.1 Dimensionality Reduction

Principal Component Analysis: An approach to cope with excessive dimensionality of the features is to combine features. A linear combination of the features can be particularly used because they are simple to compute and analytically tractable. One of the approaches to do this is Principal Component Analysis(PCA). It projects the features into a lower dimensional subspace in a way that is optimal in a sum-squared error sense. It uses the Karhunen-Loeve transformation to transform it into a lower dimensional subspace. Let μ be the mean vector and Σ the covariance matrix of the features in their original space. The eigen vectors and eigen values of the covariance matrix are calculated. Let the eigen vectors be e_1 with a corresponding eigen value λ_1, e_2 with eigen value λ_2 and so on. Choose k eigen vectors having largest eigen values where k is the dimension of the subspace that the features is being mapped to. Now we form a transformation matrix A with these eigenvectors as columns. For a new sample x, the transformed feature vector is calculated as

$$x' = A^t(x - \mu)$$

Linear Discriminant Analysis: Fisher's linear discriminant is a feature extraction method that projects high-dimensional data onto a line and performs classification in this one-dimensional space. The projection maximizes the distance between the means of the two classes while minimizing the variance within each class. This defines the Fisher criterion, which is maximized over all linear projections, w:

$$J(w) = \frac{|m_1' - m_2'|^2}{s_1'^2 + s_2'^2} \tag{2.7}$$

where m_1' and m_2' represent the projected means of both the classes, $s_1'^2$ and $s_2'^2$ represent the variances after projection. Maximizing this criterion yields a closed form solution that

involves the inverse of a covariance-like matrix. This method has strong parallels to linear perceptrons. We learn the threshold by optimizing a cost function on the training set. Let the original means be m_1 and m_2, original covariances be s_1 and s_2 respectively.

$$\begin{aligned}
(m'_1 - m'_2)^2 &= (w^t m_1 - w^t m_2)^2 \\
&= w^t (m_1 - m_2)(m_1 - m_2)^t w \\
&= w^t S_B w \\
s_i^2 &= \sum_x (w^t x - w^t m_i)^2 \\
&= \sum_x (w^t (x - m_i)(x - m_i)^t w \\
&= w^t S_i w \\
S_W &= S_1 + S_2 \\
J(w) &= \frac{|m'_1 - m'_2|^2}{s'^2_1 + s'^2_2} \\
&= \frac{w^t S_B w}{w^t S_w w}
\end{aligned}$$

S_w is a scatter matrix that encapsulates the information of within class scatter and W_B is the matrix that encapsulates the information of between class scatter. $J(w)$ is minimized when w is the eigen vector of the matrix $S_w^{-1} S_B$ with the highest eigen value.

2.4.2 Feature Subset Selection

The reduction of dimensionality is a computationally expensive process. This can be replaced by a feature subset selection methods that are available where only a few features with information required for classification are used and the remaining are ignored. The methods used are

1. In *Forward Search* one feature is taken initially and the classification performance is calculated. Each feature is added to the existing subset taken and if it results in an improvement in classification performance, it is added to the existing subset, else it is removed. This process is repeated until K such features are present in the subset. The process is repeated until the addition of any feature results in lower classification performance than that of the existing subset.

2. In *Backward Search* all the features are taken initially and the classification performance is calculated. Each feature is then removed from the existing subset and if it results in an improvement in classification performance, it is removed else it is added to the subset. The process is repeated until the removal of any feature results in lower classification performance than that of the existing subset.

3. The two previous algorithms stop the search process if the evaluation function decreases with respect to the previous step. As the evaluation function can exhibit a non-monotonic behavior, it can be effective to continue the search process even if the evaluation function is decreasing. *Tabu Search* is based on this concept.In addition, it implements both a forward and backward search strategy. The search starts from the full feature set. At each step, adding and eliminating one feature creates new subsets. Then, the subset that exhibits the highest value of the classification performance is selected to create new subsets. It should be remarked that such subsets are selected even if the classification performance is decreased with respect to the previous step. In order to avoid the creation of the same subsets in different search steps (i.e., in order to avoid cycles in the search process), a feature added or eliminated cannot be selected for insertion/deletion for a certain number of search steps. Different stop criteria can be used. For example, the search can stop after a certain number of steps, and the best subset created during the search process is returned.

2.4.3 Optical Character Recognition

In order to facilitate digital processing of the information contained in documents, they must be converted to a format understandable to a computer. Representation of documents as images is undesirable because of the large storage requirements and impossibility of many common user-level operations like editing or searching. For example, a digitized document of A4 size may take 35 MB when represented as image, and only 10KB when stored as text. In a text form, information extraction can also be attempted easily. An OCR system aims at digitizing the text in documents in a cost and labor effective manner. The primary purpose of such systems is to derive the meaning of the characters and words from their bit-mapped images. Some of the important application domains of OCR systems are processing of paper document archives, document reader systems for the visually impaired, automated applications like form processing, storage of historical vintage documents, office automation, data entry, document databases for text mining and information retrieval.

The decrease in prices of memory, storage and processor have ensured powerful computing facilities to process documents and design powerful OCR systems. New feature extraction and classification techniques have supplemented these developments. However, the design of OCR systems for Indian language scripts introduces many challenges and issues to be considered in a region-specific context.

Languages like English have characters as their building blocks. The smallest entity that can be easily extracted from a document in such languages is the character. Indian scripts have originated from the ancient Brahmi script which is phonetic in nature. Thus, in Indian languages, syllables form the basic building blocks. If all the syllables are considered as independent classes and recognized, a very large amount of information has to be stored. This creates the need for splitting the syllables into their constituent components, which

is a non-trivial task. For the wide set of languages that exist in a country like India, the current number or the variety of OCR systems are not sufficient. This motivates the need for development of a comprehensive OCR system, which can cater to current document processing needs of the Indian community.

The OCR consists of four major components, preprocessing, segmentation, script separation, feature extraction and classification. The preprocessing stage consists of binarization and skew correction of the document image. On scanning a document image, we get a monochromatic image which is binarized to form a black and white image. A skew in the image is detected and corrected in the black and white document image. Segmentation extracts components, the collection of which form characters.

A feature is defined as a numerical value describing a salient property of an entity. Features offer an alternate representation of the input data. These features form a feature vector corresponding to the entity (in this case, the character component isolated from the document image). Features are extracted from the characters isolated from the document. Initially, the bounding rectangle of each character is found, and is scaled to a standard size. This makes the features size invariant. The feature vector used in this system is obtained by adjoining all the rows of the scaled character image to form a single contiguous row. This vector consists of 1s and 0s representing foreground and background respectively. There are many efficient methods to reduce the dimensionality of the feature space by statistically analyzing their distribution or structurally analyzing their geometric properties.

Chapter 3

Design of Nodes for DDAG Classifier

Classical pattern recognition focuses on designing optimal classification algorithms for two class classification problems [19]. However most of the real-life problems like character recognition, biometrics etc. are inherently multiclass in nature and need more sophisticated classifiers. Traditionally, this has been addressed with the help of *one versus rest classifiers* integrated using a *winner takes all approach* in the case of multilayer perceptrons. Since smaller units can be designed effectively, integrating them to solve multiclass problem is a promising alternative. This has been motivated by the recent advances in classifier design procedures [16, 17, 1, 67, 29, 3, 7, 27].

The field of binary classification is mature, and provides a variety of classical approaches to solve the problem including the recent algorithms like Support Vector Machines (SVM), Adaboost, etc. The direct solution of building multiclass classifier [85] results in a complex optimization procedure [29]. most of the existing multiclass algorithms address the problem by converting it into smaller sets of binary classification problem, which are then combined using a suitable combination rule. For example, output nodes in a MLP corresponds to a binary classifier which is trained positive for samples from one class, and negative for the samples from the rest. Another popular binarization approach is to divide the problem into ${}^{N}C_2$ classification problems for each pair of classes and combine the results using voting methods. This is called the *one-vs-one approach*. Recent techniques, motivated by the error correcting codes argues for design of classifiers between two mutually exclusive sets of classes [16, 1]. They can be considered as a generalization of the existing binarization techniques.

Multiclass classifications problems are inherently difficult than binary problems, as the classification algorithm has to learn a greater number of decision boundaries. In binary classification, the biggest advantage is that, one class when considered positive, makes the other just a compliment. In this case, learning the appropriate decision boundary to one

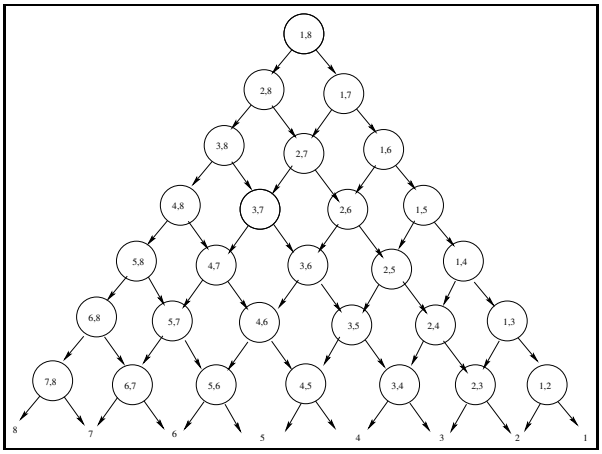

Figure 3.1: An eight class DDAG arrangement of pairwise classifiers

class is sufficient, as the other is simply the complement of this class. However, in multiclass, each class has to be explicitly defined. Errors can occur at various places in the construction of multiple boundaries, which results in overall error that is significantly greater than simple binary problems.

Popular direct multiclass approaches like Bayesian, KNN, or decision trees can be interpreted as using the class conditional densities to predict the label of a sample. However, in large dimensions, usually the number of patterns is very small, and it is very difficult to obtain accurate densities or probability models. Moreover, the problem of binary classification has been thoroughly studied theoretically and empirically, whose knowledge can be incorporated in dealing with these problems as combination of binary approaches, rather than direct multiclass problems.

3.1 DDAG as a Classifier

A DDAG classifier is a graph based combination of ${}^{N}C_2$ classifiers, as shown in the figure 3.1. A formal definition of DDAG is stated in [67] as

Definition 1: Decision DAG *Given a space X and a set of boolean functions $\mathcal{F} = \{f : X \to \{0, 1\}\}$, the class $DDAG(\mathcal{F})$ of Decision DAGs, on N classes over $\mathcal{F}$ are functions which can be implemented using a rooted binary DAG with N leaves labeled by the classes where each of the $K = N(N-1)/2$ internal nodes is labeled with an element of $\mathcal{F}$. The nodes are arranged in a triangle with the single root node at the top, two nodes in the second layer and so on until the final layer of N leaves. The $i-th$ node in $j-th$ layer $j < N$, is connected to the $i-th$ and $i+1$-st node in the $j+1$-st layer.*

Given an input sample x for classification, it is first presented to the root node of the

DDAG for decision. The decision that happens at each node of the classifiers is not which class the sample belongs to, but which class the sample *does not* belong to. One of the classes is rejected as a feasible class at the root node and the sample thus reaches the root node of the left or right sub-DAG depending on which sub-DAG does not contain the rejected class. Another class rejection takes place at this root and the sample follows an edge leading to one of the sub-DAGs of the current root node. This way, the sample selects $N-1$ nodes before it reaches the leaf node of the DAG. The sample x is assigned the label corresponding to the leaf node it reached. A DDAG architecture, with nodes as pairwise classifiers using SVM is proposed in [67]. The path taken by the sample in reaching the leaf node is called the *evaluation path*.

The multi-class classifier built using decision directed acyclic graph (DDAG) for a 8-class classification problem is shown in Figure 3.1. It can be observed that the number of binary classifiers built for a N class classification problem is $N(N-1)/2$. The input vector is presented at the root node of the DAG and moves through the DAG until it reaches the leaf node where the output (class label) is obtained.

3.1.1 Properties

Graphs are popular for modeling classifiers. Graphical models like Bayesian Nets [31] and Hidden Markov Models [19] solve the subproblems in the vertices and propagate the inferences in a stochastic framework. These classifiers can be viewed as directed or undirected graphs. Graph-based models are more popular for classifier combination, where each node is a classifier and edges become directed [67, 68]. Based on the decision at each node, a sample 'traverse' through the graph along the corresponding edge until a terminal node is reached.

A tree is a simple graph G such that there is a unique simple non-directed path between each pair of vertices of G. Decision trees are classical examples of a tree-based classifier combination scheme. In a decision tree classifier, decision is taken at each node based on some of the attributes and samples traverse down the tree along the selected path. Another possible tree-classifier is by designing the tree such that each node partitions the classes into mutually exclusive sets. A DDAG can be designed to solve the same problem in a better manner without partitioning into mutually exclusive sets of classes [67]. This can reduce the risk in making a mistake by giving too much importance to the root node, or the nodes that are encountered initially.

3.2 Character Recognition using DDAG

K-nearest neighbor and Neural Network classifiers are popular for character recognition applications, especially for Indian scripts. Superior classification is expected to warrant large computational power. Storage capacity and computational power of personal computers

Figure 3.2: Some of the Degraded Telugu characters

have exploded in the last few years and it is now possible to employ better classifiers to achieve improved performance.

Support Vector Machines [67, 64] are pair-wise discriminating classifiers with the ability to identify the decision boundary with maximal margin. Maximal margin results in better generalization – a highly desirable property for a classifier to perform well on a novel data set. From the statistical learning theory point of view, they are less complex (smaller VC dimension) and perform better (lower actual error) with limited training data. Classifiers like KNN provide excellent results (very low empirical error) on a data set on which they are trained, while the capability to generalize on a new dataset depends on the labeled samples used. Computational complexity of KNN increases with more and more labeled samples.

Identification of the optimal hyperplane for separation involves maximization of an appropriate objective function. The training process results in the identification of a set of important labeled support vectors x_i and a set of coefficients α_i. Support vectors are the samples near the decision boundary and most difficult to classify. They have class labels $y_i (i.e., \pm 1)$. The decision is made from $f(x) = sgn(\sum_{i=1}^{l} \alpha_i y_i K(x_i, x))$. The function K in the previous equation is called the kernel function. It is defined as $K(x, y) = \phi(x).\phi(y)$, where $\phi : R^d \longrightarrow H$ maps the data points in d dimensions to a higher dimensional (possibly infinite dimensional) space H. For a linear SVM, the kernel function is defined by: $K(x, y) = x.y$. We do not need to know the values of the images of the data points in H to solve the problem in H. By finding specific cases of Kernel functions, we can arrive at Neural networks or Radial Basis Functions. From the pairwise SVM classifiers, a Directed Acyclic Graph based multi-class classifier is formed [67].

We evaluated the performance of the recognition process on approximately 200000 sample characters for Telugu and Hindi. Documents were skew corrected and components were extracted. Subsets from this data set were used for training and testing in the experiments described in this section. Dimensionality reduction is made using PCA, Telugu needs around 150 features for representation, while Hindi needs only 50. An alternate algorithm for optimal feature extraction is presented in Section 3.3.

We consider two popular fonts of Hindi and Telugu languages – *Shusha* and *Nai Dunia* for Hindi, and *TL-TT Hemalatha* and *TL-TT-Harshapriya* for Telugu. We consider an average of 20000 samples for the experiment. Scale and resolution of characters usually influence the performance of the system. In our experiment, we also considered samples printed at various sizes and scanned at various scales. This resulted in an average of 30000 labeled

Expt Details	Language	Font	KNNI	KNNP	SVMLI	SVMLP	SVMQ
Homogeneous Data	Hindi	Shusha	99.60	99.16	99.64	99.25	99.36
	Hindi	NaiDunia	99.51	99.20	99.86	99.13	98.96
	Telugu	Hemalatha	98.67	95.24	98.80	98.66	98.85
	Telugu	Harshapriya	99.30	99.03	99.25	99.67	99.02
Scale Independence	Hindi	Shusha	96.61	95.78	98.12	97.67	97.69
	Hindi	NaiDunia	96.11	94.30	97.00	97.12	97.17
	Telugu	Hemalatha	97.55	94.97	97.31	97.32	97.10
	Telugu	Harshapriya	96.22	92.73	97.12	97.22	97.13
Degraded Data	Hindi	Shusha	93.03	92.51	96.20	96.92	96.95
	Hindi	NaiDunia	94.60	93.40	96.50	96.98	97.06
	Telugu	Hemalatha	93.03	93.31	97.23	97.92	97.95
	Telugu	Harshapriya	93.10	93.93	97.14	97.83	97.72
Font Independence	Hindi	N. A	92.64	92.97	97.10	96.68	96.72
	Telugu	N. A	93.03	92.30	97.87	97.62	97.58

Table 3.1: Recognition accuracies of the system on synthetic and scanned Indian language character data. The SVM results are shown for linear and quadratic kernels. KNN and SVM are the classifiers, I and P are the Image and PCA features, L and Q are the linear and quadratic kernels for SVM.

samples. Results are shown in Table 3.1. Results show that for the homogeneous data, recognition is very good for all combinations.

Characters in a document image are usually degraded in quality. We use the degradation model given in [47] to simulate the distortions due to the imperfections in scanning. Black and white pixels are swapped according to some probabilities directly related to the distance from the boundary. Some of the degraded Telugu characters are shown in Figure 3.2. The results of degradation are given in Table 3.1. It can be observed that the degradation is more serious for Telugu than Hindi, for a fixed dimensional representation.

We also tried to see the performance of KNN classifier so as to make a comparison with SVM. KNN based classification results were computed for many values of K; $K = 5$ is shown in Table 3.1.

The over-all performance of the system is 96.7%. Misclassifications are present due to the distortions created by the skew correction. SVM-based classifiers are found to perform better than KNN. This result is better than the accuracies reported in [4] on smaller or unknown data set.

The paper and printing qualities were good for all these experiments. These results, as such, cannot be expected for newspaper and other real-life documents. Presently, we are

working on algorithms to achieve good performance on real-life poor quality documents.

Support Vector Machines (SVMs) provide better results than KNN classifiers. Though we tried many kernels for the SVMs, we could not find any considerable difference between their performances. We use linear kernels for our applications. From the previous experiment, we find that the result of the SVM-PCA combination is computationally affordable and performance-wise superior. The proposed OCR has tremendous scope for further improvement. Some of the works done in this direction are discussed in the next section.

There are a number of applications where a smaller number of features will be preferable. We report the result of our work in finding a set of optimal discriminant vectors for the pairwise classification using SVMs (refer Section 3.3 for details).

3.3 Node-level Design and Features Selection

In a DDAG, all the classifiers are trained on individual pairs of classifiers, and hence may use different feature representations. Using pair-specific features is more suitable for the DDAG than using features derived from all the classes globally. Usually, features like PCA are used with DDAGs, and they are computed by considering all the classes at once. This has the following problems

- Representing multiple classes is difficult than representing two classes. Global feature extraction methods like PCA aim at obtaining features which best represent each class for classification. However, the representation which the pairwise classifier looks for, is to classify between two classes it would be trained upon. Considering multiple classes in the feature extraction phase of each classifier makes the problem of representing the current class difficult as the representation has to be unique with respect to more than one set of samples. Where as, using a pairwise feature extraction method, results in a feature representation which has to be unique with respect to only one other class, and hence can be derived efficiently and with simple methods.

- The PCA features are more representative than discriminative and hence are less suitable for classification tasks. Features obtained using Linear Discriminant Analysis (LDA) methods like Fisher discriminants, Optimal Discriminant Vectors of Foley-Sammon [23] are advantageous in these situations.

Considering these two factors, we use a pairwise LDA feature extraction method, independently at each pairwise node of the DDAG, after reducing dimensionality of the feature vector using PCA.

The objective of Linear Discriminant Analysis (LDA) is to find a projection, $y = Dx$ (where x is the input and D is the transformation), that maximizes the ratio of the between -class scatter and the within-class scatter [19]. For these, we apply the algorithm originally

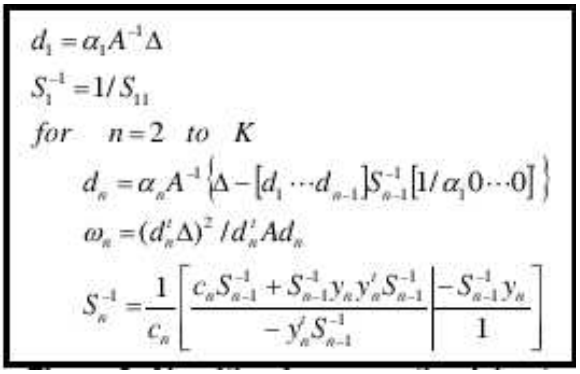

Figure 3.3: Algorithm for Computing Optimal Discriminant Features

proposed by Foley and Sammon [23]. The algorithm extracts a set of optimal discriminant features for a two-class problem which suits the Support Vector Machine (SVM) classifier.

Consider the ith image sample represented as an M dimensional (column) vector x_i , where M is the reduced dimension using PCA. For the sets of training samples, x_1, L, x_N , we compute with-in class scatter (W) matrix and between-class difference (δ) as:

$$W_i = \sum_{j=1}^{N_i} (x_{ij} - \mu_i)(x_{ij} - \mu_i)^t$$

$$\delta = \mu_1 - \mu_2$$

where x_{ij} is the jth sample in ith class.

Some of the with-in-class scatter is also determined by:$A = cW_1 + (1-c)W_2$, where $0 \leq c \leq 1$ and the scatter space using $S_{ij} = d_i A^{-1} d_j$. The algorithm presented in Figure 3.3 is used for extracting an optimal set of discriminant vectors (d_n) that corresponds to the first$L$ highest discriminant values such that ($\omega_1 \geq \omega_2 \geq ... \geq \omega_L \geq 0$) [23]. Here K is the number of iterations for computing discriminant vectors and discriminant values, $c_n = s_{nn} - y_n^t S_{n-1}^{-1} y_n$, $y_n = [S_{in} ... S_{(n-1)(n)}]$, and α_n is chosen such that $d_n^t d_n = 1$.

3.3.1 Performance Analysis

Principal Component Analysis (PCA) yields projection directions that maximize the total scatter across all classes. In choosing the projection which maximizes total scatter, PCA retains not only between-class scatter, that is useful for classification, but also within-class scatter, that is unnecessary for classification purposes.

For a given pair of classes, Linear discriminant analysis (LDA) attempts to maximize the separation between the classes after the projection. Usually this is done as the maximization of between class scatter and minimization of between class scatter simultaneously. There are different ways to perform an LDA, of which an optimal method is proposed by Foley

Dataset	Normal features	PCA	LDA
Malayalam	99.7	99.7	100
Vowel	55.31	53.19	53.19
Satimage	88.03	87.58	90.12
Pendigits	95.77	95.46	95.99
Letter	72.80	75.16	81.20
Optdigits	97.90	97.90	98.63

Table 3.2: Performance Comparison of Normal, PCA and LDA features classified using a DDAG

and Sammon called Optimal Discriminant Vectors (ODV), which we implemented in our work.

To investigate the performance of PCA and LDA on the various dataset, we undertake experimental analysis on both techniques. Table 3.2 shows the test results obtained. LDA is observed to show better results than PCA on almost all the datasets.

3.4 Reduction in Storage Complexity

For a multiclass problem, when a classifier like DDAG is being used, each pairwise classifier is independent of each other. The feature representation of the sample can be independent for each of these classifiers. Also, using a single set of features for all the samples may not be good enough, if the features are globally extracted. This suggests that, using a linear discriminant for each pair of classes in each pairwise classification is an attractive solution to design better classifiers. However, the only disadvantage of such an approach is that the classification becomes slower, as the features are to be extracted at each step, which involves a transformation of input vector using a matrix multiplication. This operation has to be performed $O(N^2)$ times, and is expensive as the number of classes increase.

If there are large number of classes, it is usually the case that a large number of discriminant vectors are oriented in parallel, or close to parallel directions to each other. With a negligible loss in representation or discrimination quality, the set of similarly oriented vectors can be represented using a single vector. All these pairwise optimal discriminant vectors can be compressed into a set of basic vectors representing all the major projection directions in the feature space. This greatly reduces the problem of storage complexity.

We achieve this compression using PCA. The set of LDA vectors for each pair are stacked onto to a single matrix. This matrix is rank deficient. The rank of the matrix gives an idea about the number of independent projection directions of the optimal discriminant vectors. After performing the PCA, the first few non-negative eigen values are chosen as the projection directions. These are the most frequent projections existing in the LDA

Dataset	Original	PCA	PCA+LDA	LDA+PCA
Satimage	19	17	75	11
Optdigits	64	41	225	36
Pendigits	16	13	225	11
Vowel	10	9	275	9
Letter	16	15	1625	14

Table 3.3: Number of features in the original dataset, after PCA, PCA followed by LDA and LDA followed by PCA

Dataset	PCA + LDA	LDA + PCA
Malayalam	100	100
Vowel	53.19	53.19
Satimage	89.43	89.31
Pendigits	95.99	95.99
Letter	80.40	80.40
Optdigits	98.46	98.34

Table 3.4: Performance Results of LDA followed by PCA and vice versa.

matrices for all the pairwise feature transformations. This set of eigen vectors is used as a final projection matrix for obtaining global features for the complete DDAG. The set of features obtained by this approach has at least few features that are optimal for all the pairwise classifiers in the DDAG. The results obtained using this approach are summarized in Table 3.4. Experiments are conducted to study the space and time complexities, and accuracies of the LDA followed by PCA, and PCA followed by LDA. It is observed that there is not much difference in both the approaches in accuracy, however, LDA followed by PCA is found to use very less number of features when compared to PCA followed by LDA, with no or minimal loss in the accuracy. The number of features used by both approaches are summarized in Table 3.3. It is obvious from the table that the difference in the number of features is extremely large, and LDA followed by PCA is better.

3.5 Boosting DDAGs

The performance of a multiclass classifier f built using combination of multiple binary classifiers depends on three parameters – the feature representation used F, the set of the binary classifiers used C and the architecture A used to combine the individual classifiers. The multiclass classifier $f(F, C, A)$, has to be built specifically to each problem, which

requires a careful feature selection, classifier selection and architecture selection. However, most of the usual approaches have a predetermined F, C and A. In this work we propose an approach called Boosting to boost the performance of weak nodes and architecture of the DDAG.

Boosting [27] is a popular approach for improving the performance of a weak learner [26, 75]. Boosting iteratively applies to a weak classification algorithm to a subset of the data selected based on the weights $w_i, i = 1 \cdots N$, defined on the samples, which are updated after each iteration. The weak learner has to find a hypothesis from the training set which is at least better than random. These hypothesis are then combined using a weighted averaging of the decisions of the individual hypotheses.

Boosting is an iterative procedure, where the newer classifiers are influenced by the performance of the previously built classifiers. Each new model being built attempts to become an expert for the instances misclassified by the earlier classifiers. This makes the classifiers provide information that is complimentary to the rest of the classifiers.

The first provable polynomial time boosting algorithm was published in [74]. A much more efficient algorithm called the Adaboost is later proposed in [27, 25].

For a given dataset D with N sample points $\{x_1, x_2, \cdots, x_n\}$, Initialize the weights, $w_i, i = 1 \cdots N$ For $m = 1$ to M,

1. Build the classifier f_m for the current dataset
2. Compute the error of the classifier m, e_m on the dataset using $e_m = \frac{\sum_i w_i^{(m)} I(y_i \neq f_m(x_i))}{\sum_i w_i^{(m)}}$
3. Compute the weights of the classifiers $\alpha_m = \frac{1}{2} \log \frac{1-e_m}{e_m}$
4. Compute the weights of the data samples $w_i^{(m+1)} = w_i^{(m)} e^{\alpha_m y_i f_m(x_i)}, i = 1 \cdots N$

Let $f_1(x), f_2(x) ... f_M(x)$ represent the M classifiers that are built using a boosting sequence, the final classifier is obtained as a weighted sum of these individual classifiers.

$$f_{boost}(x) = \sum_{i=1}^{M} \alpha_m f_i(x)$$

In Practice, AdaBoost was observed to be superior to many other algorithms in terms of efficiency and computational complexity. Adaboost builds over a set of weak classifiers, and the only programming involved is to redefine the weights over the samples of datasets in each iteration which makes it easy to program, and while execution it is fast because it calls the weak learner some n number of times, and combines their decision through voting. The algorithm Adaboost is called so because of its adaptive nature. No parameters or priors about the weak learners have to be set before for the ensemble to work better. The algorithm adapts itself in each iteration, whatever may be the underlying weak learner

Dataset	Normal features & Boosting	LDA + PCA & Boosting
Satimage	83.83	84.18
Optdigits	98.56	98.92
Pendigits	96.42	97.24
Vowel	60.90	75.56
Letter	76.15	81.26

Table 3.5: Recognition Accuracies of Boosted DDAG

used, although the final performance of the algorithm does depend on the problem, data and the weak learner.

The main idea behind node boosting is that nodes which result in misclassifications by the overall DDAG have to be improved in order for DDAG to perform well. This involves two steps - identifying the under performing nodes, and improving them, using boosting. Identifying the nodes to be improved can be done using simple tests on the generalization performance of each node when it is trained. A k-fold cross validation approach is used to estimate the generalization error of a node, and if it is below a specified acceptability level, it is selected for boosting.

The Node level boosting methods are used with an optimal feature extraction algorithm to build a high-performance OCR system. Table 3.5 presents test results before and after boosting the DDAG. This is done after extracting features using LDA and then followed by PCA for dimensionality reduction. The result shows that there is relative improvement in performance after applying boosting techniques in most of the datasets.

3.6 Summary

An optimal feature extraction algorithm and algorithm to improve the DDAG classifier are proposed. Node-level design and features selection enables to extract discriminant features for pair-wise class combinations that better suits the SVM classifier were used to achieve better classification rates. We extracted features using LDA followed by PCA for dimensionality reduction. This setup solves the problem of space complexity encountered by using PCA followed by LDA. Boosting the DDAG at node level is also presented. With this, we registered high accuracy with less space and time complexity. However, this work addresses boosting DDAG at node level only. Arrangement of the nodes in the DDAG is observed to have an effect on the performance of the complete multiclass classifier. So there is a need to think of design issues in the DDAG architecture.

Chapter 4

Design of Class Order for DDAG Classifier

The field of binary classification is mature, and provides a variety of classical approaches to solve the problem including the recent algorithms like Support Vector Machines (SVM), Adaboost, etc. The direct solution of building multiclass classifier [85] results in a complex optimization procedure [29]. Hence most of the existing multiclass algorithms address the problem by converting it into smaller sets of binary classification problem, which are then combined using a suitable combination rule. For example, output nodes in a MLP corresponds to a binary classifier which is trained positive for samples from one class, and negative for the samples from the rest. Another popular binarization approach is to divide the problem into ${}^{N}C_{2}$ classification problems for each pair of classes and combine the results using voting methods. This is called the *one-vs-one approach*. Recent techniques, motivated by the error correcting codes argues for design of classifiers between two mutually exclusive sets of classes [16, 1]. They can be considered as a generalization of the existing binarization techniques.

Graphs and Trees are effective means for integrating multiple classifiers [68, 67, 3, 31]. In general, nodes are classifiers and directed edges provide the path along which evaluations are done, until a terminating node is reached. Decision trees are popular in many data mining applications. Designing decision trees is an NP-Hard problem and greedy algorithms are popular for its design, though they are non-optimal. A Decision Directed Acyclic Graph (DDAG) is a generalization of decision tree. It is used to combine pairwise SVMs in [67]. More specifically, a rooted binary DDAG is used, where each node is a pairwise classifier and has a degree 2. An example DDAG for six classes is shown in Figure 3.1. Each node has two edges entering into it, and two edges emanating from it. All the edges are directed, and are used in selecting the next classifier to be used to classify the sample.

A graph-based classifier has many advantages. There are various procedures followed for integrating the base classifiers. All these approaches contain a set of classifiers, and a set

of rules which form the combination mechanism. These set of classifiers and combination rules, suggest that the combined classifier can be represented as a graph. A graph based representation supports the notion of modularity, which says that complex systems are built by combining simpler parts. Using Graph based representation allows intuitive interpretations to be made about the decision. It is possible to extend the theoretical analysis and design techniques available for two class classifiers. Algorithms for efficient traversal can be built in graphs, avoiding using the whole set of classifiers to come up with a decision. Node design, and graph design algorithms can be built. The graph based representation also serves as a common underlying formalism, allowing us to view all existing combination mechanisms as instances of it.

The design issues in the DDAG architecture are addressed. Arrangement of the nodes in the DDAG is observed to have an effect on the performance of the complete multiclass classifier. Also the prior probability information available from the dataset is used in the formulation, which was not the case in previous DDAG approaches. The problem of finding the best arrangement of nodes is formulated in terms of minimizing the error rate of the complete classifier. Algorithms for the design of the classifier under various assumptions on the classification probability and priors on the classes are presented. Since the problem belongs to the class of permutation problems, the problem is shown to be an NP-complete problem, and the proof is presented.

4.1 Class Ordering for Design of the DDAG

4.1.1 Graph based Classifier Combinations

Graphs are popular for modeling classifiers. Graphical models like Bayesian Nets [31] and Hidden Markov Models [19] solve the subproblems in the vertices and propagate the inferences in a stochastic framework. These classifiers can be viewed as directed or undirected graphs. Graph-based models are more popular for classifier combination, where each node is a classifier and edges become directed [67, 68]. Based on the decision at each node, a sample 'traverse' through the graph along the corresponding edge until a terminal node is reached.

A tree is a simple graph G such that there is a unique simple non-directed path between each pair of vertices of G. Decision trees are classical examples of a tree-based classifier combination scheme. In a decision tree classifier, decision is taken at each node based on some of the attributes and samples traverse down the tree along the selected path. Another possible tree-classifier is by designing the tree such that each node partitions the classes into mutually exclusive sets. A DDAG can be designed to solve the same problem in a better manner without partitioning into mutually exclusive sets of classes [67]. This can reduce the risk in making a mistake by giving too much importance to the root node, or the nodes that are encountered initially.

A DDAG is a computationally attractive proposition to many existing integration schemes. It employs considerably less number of classifiers to test a new sample when compared to other methods. A majority voting formulation based on pairwise classifiers needs an order higher number of classifiers to take the final decision. Training and Testing complexities in terms of number of classes (N) and number of samples per class (M) is summarized in Table 4.1.

Architecture	N	Training Complexity	Testing Complexity
One-Vs-Rest	N	Nf(M)	N
Majority Voting (1v1)	N(N-1)/2	$\frac{1}{2}N(N-1)f(2M/N)$	N(N-1)/2
DDAG	N(N-1)/2	$\frac{1}{2}N(N-1)f(2M/N)$	N-1

Table 4.1: Comparison of the complexities of various classifiers

4.2 Design Under Restricted Probabilistic Setting

4.2.1 Objective of the Design

Success of the classifier combinations depends on the design strategy it employs. For example, the relative placements of the nodes in the DDAG (shown in Figure 3.1) is important for improving the overall classifier performance. Design of Tree-based classifiers have received much of attention in literature. Decision trees are designed using greedy, lookahead and backtracking approaches [68]. Objective in this case is to design a tree of minimal size with least training error.

In the case of DDAG, since the architecture is fixed, the design problem reduces to that of an optimal arrangement of nodes to maximize the classier accuracy. Consider an N class classification problem. Let $P_i, 1 < i < N$ be the apriori probabilities of the classes ω_i and Q_i be the probability that a sample belonging to ω_i is misclassified into any other class by the overall classifier. The notation Q_i is used for misclassification probability of the multiclass classifier on class i, The misclassification probability of a specific configuration A of a multiclass classifier can be expressed as

$$J(A) = \sum_{i=1}^{N} P_i Q_i$$

In the above equation, the misclassification probability of a class by the multiclass classifier depends on the way individual classifiers are combined. The objective is to find an arrangement A for which the overall misclassification probability is the least. This analysis incorporates the apriori probabilities of the classes in the analysis.

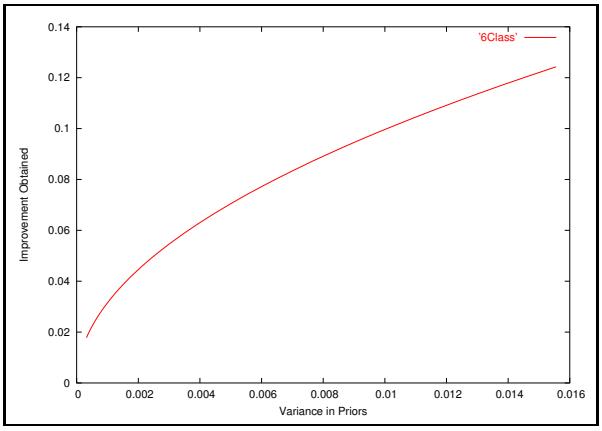

Figure 4.1: Variation of improvement in performance of DAG classifiers with variance of apriori probabilities of the classes

4.2.2 Designing the DDAG

A DDAG classifier for N classes can be built with the help of $N(N-1)/2$ classifiers as described in the previous section. However, with different arrangements, the performance of the classifier-ensemble can vary a lot. A critical question in DDAG classifier is the arrangements of nodes to maximize the performance. We define a *relevant classifier* to a class ω_i as the classifier which is trained to correctly classify samples from ω_i against any other class ω_j $j \neq i$. Let the misclassification probability of relevant classifier be q. Nodes in a DDAG are arranges such that for samples which are not relevant to the classifier, its decision will not be critical. For each class there are a $N-1$ relevant classifiers in a DDAG. Each sample $\mathbf{x}$ presented to a DDAG classifier follows a path, decided by the decisions at individual node until it reaches a leaf node. This path is referred to as *evaluation path*. For samples from any class, evaluation path is of length $N-1$, where all the nodes along the evaluation path need not be a relevant classifier. However, during the traversal, once a relevant classifier is reached, all subsequent nodes will be a relevant classifier for the sample to be correctly classified. If any of these series of relevant classifiers, make a mistake, sample gets misclassified. In other words, the unanimous decision of these classifiers is critical to the correct classification. It can be easily seen that the number of relevant classifiers along an evaluation path may vary for different classes. This variation can be effectively used to improve the performance of the classifier-ensemble. We define the number of relevant classifiers in an evaluation path as *relevant path length*. For example, consider a problem of designing a multiclass classifier from pair wise classifiers for 4 classes. We can have 4! ways of arranging the classes in the leaf nodes. Given the arrangement of classes in the leaf nodes, indirectly define the entire DAG. It is observed that the DAGs differ in their performance. To study the variation of performance with variations in apriori probabilities, experiments were conducted. Difference in performance between the best possible and worst possible

DAGs is computed for each experiment. The performance improvement becomes significant more and more with the variance in apriori probabilities of the classes. In Figure 4.1, we show the the improvement in DAG performance with respect to the variance in apriori probabilities of the classes. When apriori probabilities are equal, all possible DAGs result in same performance. However, as the variance increase, performance gain increases. The problem of identification of the best DDAG out of the $N!$ possible ones can be solved in polynomial time in this case, as we demonstrate below.

For the sample DDAG shown in Figure 3.1, a sample from class ω_6 can get correctly classified only if 5 classifiers $((1-6) \Rightarrow (2-6) \Rightarrow (3-6) \Rightarrow (4-6) \Rightarrow (5-6))$ correctly classifies a sample. For this sample, evaluation path and relevant path are same and of length 5. For a sample from class ω_2, the decision at the root node does not really matter. It can take many paths. For example for the path $(1-6) \Rightarrow (2-6) \Rightarrow (2,5) \Rightarrow (2,4) \Rightarrow (2,3)$, evaluation path is of length 5 and relevant path is of length 4. The same sample could take a path $(1,6) \Rightarrow (1,5) \Rightarrow (1,4) \Rightarrow (2,4) \Rightarrow (2,3)$. In this case, evaluation path remains as 5 and relevant path is of length 2. For this sample there are also paths with relevant lengths 3, and 1. In fact, for a sample from class 2, there exists one path each of relevant lengths 4, 3 and 2 and two paths of relevant length 1. The rth leaf can be reached through ${}^{N-1}C_r$ paths. In general, a node l in mth layer can be reached in ${}^{m-1}C_{l-1}$.

Accuracy of the DDAG classifier-ensemble to a given sample can be computed as the product of the accuracies of the relevant classifiers in the evaluation path. We consider that nodes which are not relevant in an evaluation path takes decisions at random with equal probability. If there are l relevant classifiers in an evaluation path and such N_l evaluation paths exist to reach a specific leaf node, The accuracy over samples taking these paths is given by.

$$N_l(\frac{1}{2})^{N-1-l}(1-q)^l$$

For the first and last leaf nodes, all the $N-1$ classifiers are relevant. For the rth leaf, the number of relevant classifiers can assume a value from $\mathcal{L} = \{1 \ldots \max(N-r, r-1)\}$ The average relevant path length are shorter for classes in the center. For the classes on the extreme, l is $N-1$ and N_l is 1. For all other classes, N_l.

The number of paths of *at least* relevant length l at rth leaf node is equivalent to the number of ways in which one can reach $r-l$th and rth node at depth $N-l$. i.e.,

$$N_l^{r\prime} = {}^{N-l-1}C_{r-l-1} + {}^{N-l-1}C_{r-1}$$

And number of paths of length l,

$$N_l^r = N_l^{r\prime} - N_{l+1}^{r\prime}$$

The accuracy of class ω_r, is given by

$$(1-Q_r) = \frac{1}{{}^{N-1}C_r} \sum_{l \in \mathcal{L}} N_l^r \left((\frac{1}{2})^{N-1-l}(1-q)^l \right) \tag{4.1}$$

The problem becomes maximization of

$$J = \sum_{i=1}^{N} P_i(1 - Q_i)$$

This function can be maximized by choosing two classes with largest P_i at each step and giving them the least available relevant path lengths. It can be observed from the following lemmas that if relevant path length increases, Q_i increases. An algorithm for designing the DDAG in this special case is presented below, followed by the proof that this actually maximizes the classification probability.

Algorithm 1 DAG1

Input: Classes ω_i, Apriori ProbabilitiesP_i

Output: Outputs a DDAG with maximum classification probability as an ordered tuple S

$S = ()$
while ω is not empty **do**
  Select two classes $\omega_i, \omega_j$with highest aprioris
  $S = (\omega_i, S, \omega_j)$
  $\omega = \omega - \{\omega_i, \omega_j\}$
end while
output S

4.2.3 Analysis of the Algorithm

Lemma 1 In a DDAG, the classes with maximum relevant path length greater, have a greater chance of misclassification.

Proof A leaf node in a DDAG can be reached in different ways, each of which has different path lengths. Let x, y be maximum relevant path lengths of two classes ω_i, ω_j, and if

$$\max(N - i, i - 1) > \max(N - j, j - 1) \tag{4.2}$$

then $Q_i > Q_j$.

Without loss of generality, assume that the $x = y + 1$. The lemma is proved by showing that $Q_i - Q_j$ is positive in this case. $Q_i - Q_j =$

$$\frac{1}{2^N} \sum_{j=1}^{y} \frac{R(y+1, j)(1-q)^j}{2^{N-j-1}} - \frac{1}{2^N} \sum_{j=1}^{y-1} \frac{R(y, j)(1-q)^j}{2^{N-j-1}}$$

$$= R(y+1,y)\frac{(1-q)^y}{2^{2N-y-1}} + \sum_{j=1}^{y-1} \frac{(R(y+1,j) - R(y,j))(1-q)^j}{2^{2N-j-1}}$$

Since $R(y+1,j) - R(y,j)$ is positive when condition 4.2 is satisfied, $Q_i - Q_j$ is positive.

Lemma 2 For a DDAG, there exists an optimal class order, where classes x, y of least apriori probability have the highest relevant path length.

Proof Let Y be the class order, where the classes a, b have the highest relevant path length, and let $P_b > P_a > P_y > P_x$. Let Y' be a solution where the classes a, b have a lesser path length than the classes x, y.

$$\begin{aligned}
& J(Y) = (P_b + P_a)Q_n + (Py + Px)Q_{n-1} \\
& J(Y') = (P_y + P_x)Q_n + (Pb + Pa)Q_{n-1} \\
& J(Y) - J(Y') \\
= \; & (P_b + P_a)(Q_n - Q_{n-1}) + (Py + Px)(Q_{n-1} - Q_n) \\
= \; & ((P_b + P_a) - (Py + Px))(Q_{n-1} - Q_n) \leq 0
\end{aligned}$$

This is always negative because $(P_b + P_a) - (P_y + P_x)$ is positive from the assumption, and $Q_{n-1} - Q_n$ is negative from lemma 2. Clearly, $(P_y + P_x) - (P_b + P_a)$ is negative, from the assumption. This means, $J(Y) < J(Y')$. This shows that Y is a better solution than Y'.

Discussion Let the last two classes, i.e the ones with highest probabilities be removed. The remaining order Y' is optimal for the remaining classes. In the algorithm, pairs are being selected from a sorted set. Even if a pair, is removed the tree still remains optimal, because the remaining class order is sorted on probabilities. This shows that the principle of optimality hods in this case and the algorithm results in an optimal solution.

Example Consider a four class classification problem. Let P={0.3,0.1,0.2,0.4} be the apriori probabilities of the classes $\omega = \{\omega_1, \omega_2, \omega_3, \omega_4\}$. Let q_{ij} represent the misclassification probabilities of the pairwise classifiers built between the pair of classes (ω_i, ω_j). Assuming that the misclassification probability of all the classifiers is same, let $q = 0.1$. Considering an arbitrary DAG with the given class order $\{\omega_1, \omega_2, \omega_3, \omega_4\}$, computing the misclassification probability using the Lemma 1, gives 0.8028. Using the algorithm $DDAG1$, if the optimal DDAG is designed by placing the more frequent classes (ω_1, ω_4) in the center, optimal DDAG order for this should be $\{\omega_1, \omega_4, \omega_3, \omega_2\}$. It can be observed that this ordering has the least misclassification probability when compared to others. The misclassification probability of the DDAG with the class order $\{\omega_1, \omega_4, \omega_3, \omega_2\}$ is 0.8892. Therefore, by redesigning the DDAG, a reduction of 43.8% in the error rate is obtained.

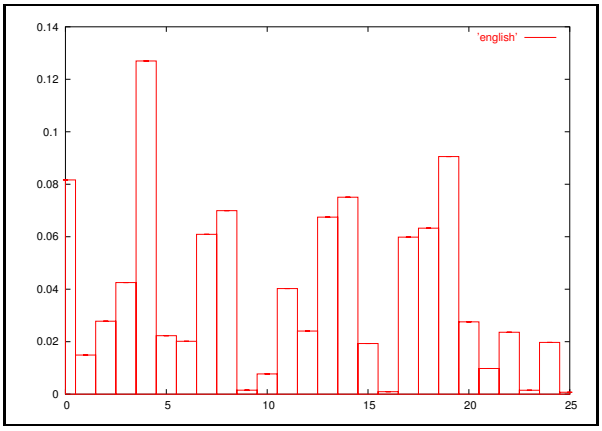

Figure 4.2: Priors of classes for the English letter dataset from the UCI repository

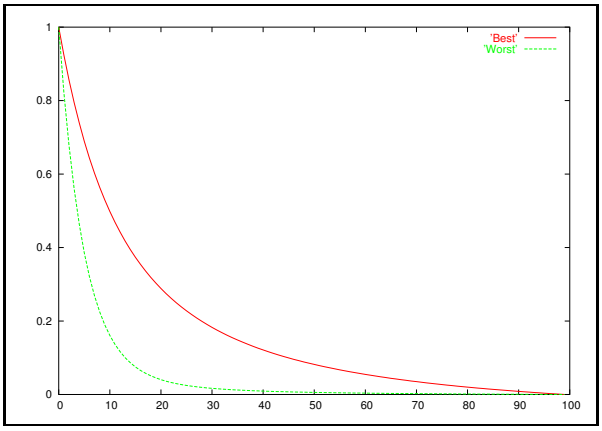

Figure 4.3: The improvement obtained using the DAG redesign algorithm on English alphabet, red line shows the improved DAG, and green line shows the worst possible DAG.

Classification of Characters Character recognition is a classical multiclass classification problem. A DDAG is possibly an effective solution to be used as a classifier for character recognition, because of the its low testing complexity. However, any arbitrary DDAG may not be the best possible classifier, as seen in the previous example. A best possible DDAG can be designed for solving the problem using the above algorithm. This improved DDAG performs better than any other possible arbitrary DDAGs. The effect of the redesigning algorithm on improvement of performance of the character recognition systems is studied using a simulation. In this simulation the priors of the alphabet are computed from a large text corpus (shown in Figure 4.2) and the misclassification probability of all the pairwise classifiers involved in the DDAG is considered to be the same. The overall performance of the classifier can now be computed using the equation 4.4. The simulation involved computing the performances of the best possible DAG formed using the above algorithm and a worst

DDAG possible with those classes. The worst possible DAG was constructed using an inverse algorithm, where the highest probability classes are placed at the ends of the DAG where as the low probability classes occur at the center. Figure 4.3 shows the performance of the best and worst possible DAGs, varying with the misclassification probability. It can be observed that in all the cases, i.e for any misclassification probabilities of the component classifiers, there is a definite positive improvement in the performance of the classifier when it is designed using the above algorithm. Also, it can be seen that if the accuracies of the component classifiers are too high or too low, the redesigning algorithm may not yield very high improvements, where as for classifiers with moderately good performance, which is usually the case, redesigning the DDAG proves to be very effective.

4.3 Design of an Optimal DDAG is NP Hard

4.3.1 Objective of the Design

The previous section discusses a design algorithm for a DDAG, with certain assumptions like the misclassification probabilities of all the pairwise classifiers are equal and symmetric, and also each classifier splits an irrelevant sample equally to take two different sub-DAGs, i.e the probability of taking either of the sub-DAGs of any root is the same, which is equal to 0.5. Although these assumptions simplify the problem to a great extent, and give an insight into the analysis of the problem, practical problems where these assumptions hold are almost non-existent. Real life pattern recognition problems are more complex, and hence require a more comprehensive problem formulation and a solution.

This section describes a general setting and attempts to provide an algorithm to design a DDAG to solve the problem efficiently. The above assumptions have to be modified so that they suit the problem in the general setting. In a general situation, misclassification probabilities are not equal and the design of the multiclass classifier becomes more difficult. The problem formulation is described in the subsequent paragraphs.

Given an N class classification problem, with priors $P = \{P_i | i = 1 \cdots N\}$ which is being solved using a DDAG built from pairwise classifiers C_{ij}, for each pair ω_i, ω_j. Let q_{ij} represent the probability that a sample ω_i is misclassified into ω_j. Note that given a pairwise classifier between two classes ω_i and ω_j, the misclassification of ω_i into ω_j need not be same as the misclassification of ω_j to ω_i, which means $q_{ij} \neq q_{ji}$. This misclassification probability applies only to the samples which are relevant to the classifier, whereas in a DDAG a classifier may receive irrelevant samples. A similar term to q_{ij} has to be defined for each pairwise classifier, which measures the number of times each sub-DAG, either left or right, is chosen when the samples are presented to a root node. This term is called the splitting factor and is represented using s^{ij}_{ki}. The subscript ij stands for the pairwise classifier the splitting factor corresponds to, and the superscript ki describes the class ω_k which is getting misclassified as ω_i. For each class ω_k, and for each pairwise classifier C_{ij},

there exist a pair of splitting factors deciding which path the class takes. It can be observed that, $s_{ij}^{ki} + s_{ij}^{kj} = 1$, as each sample belonging to a class ω_k is classified as either ω_i or ω_j by the classifier C_{ij}. Also $s_{ij}^{ij} = q_{ij}$ and $s_{ij}^{ji} = q_{ji}$. In further discussion, only s is used and it has to be understood as the misclassification probability as and when appropriate. In figure 3.1, when a sample belonging to an unknown class k, is presented to the root node C_{ij} (which is C_{16} in this case), it is classified into either of the classes ω_i with a probability s_{ij}^{ki}. The probability that the root node of the DDAG gets a sample from class k for classification depends on the prior of the class k, which is P_k. This is true for the root node of the DAG, but when a sub-DAG is selected, the sample is presented to the root node of that sub-DAG. The probability that the root node of the sub-DAG gets a sample belonging to class k to classify is not P_k, but a relative probability recomputed on all the samples the sub-DAG is defined for. Let this relative prior probability be represented as ρ_{ij}^k, where the subscript ij stands for the classifier acting as the root node of the sub-DAG and k stands for the class for which the relative prior is being defined. For a general node C_{ij}, the probability that it classifies a sample belonging to class ω_k into ω_i is $\rho_{ij}^k s_{ij}^{ki}$ and into class ω_j is $\rho_{ij}^k s_{ij}^{kj}$.

It can be observed that the DDAG discussed till now as a graph, is uniquely defined by the order of the leaf nodes. This fact can be used to form a convenient representation of the DDAG as a list of the classes in the leaf nodes. Let ϕ denote the sequence of the leaf nodes, and ϕ_{lr} represent a subsequence of ϕ with l as the left index and r as the right index. The length of the sequence ϕ is same as the number of classes. Using this notation the DDAG problem can now be formulated as a problem of finding out the best possible sequence of classes maximizing the overall classification probability of the DDAG. The accuracy of the sequence $\phi_{i,j}$, represented as $J(\phi_{i,j})$ can be written as

$$J(\phi_{i,j}) = \sum_{k \in \phi_{i,j}} \rho_{ij}^k s_{ij}^{ki} J(\phi_{l,r-1}) + \sum_{k \in \phi_{i,j}} \rho_{ij}^k s_{ij}^{kj} J(\phi_{i+1,j}) \tag{4.3}$$

and the termination condition for recursion is given when the last classifier in the series is reached, when $i = j - 1$.

$$J(\phi_{i,j}) = \rho_{ij}^l s_{ij}^{ii} + \rho_{ij}^r s_{ij}^{jj}$$

where $\sum_{k \in \phi_{i,j-1}} \rho_{ij}^k s_{ij}^{ik}$ represents the probability of choosing the left sub-DAG and $J(\phi_{l,r-1})$ stands for the accuracy of the left sub-DAG. Similarly, the second term in the expression corresponds to the product of probability of choosing the right sub-DAG and the accuracy of the right sub-DAG. $J(\phi_{1,N})$ represents the accuracy of the complete DAG. The objective is to design the sequence ϕ in such a way that the accuracy of the overall DAG, $J(\phi_{1,N})$ is maximized. Let ϕ^* represent the optimal sequence, it can be obtained as

$$\phi^* = \arg\max_{\phi} J(\phi_{1,N})$$

4.3.2 Branch and Bound Algorithm

The problem presented here is a combinatorial optimization problem, where one of the $n!$ possible sequences is to be selected. As the number of classes increases, as in the case of character recognition, the time taken to design the DAG becomes intractable. A Branch and bound based algorithm is presented to reduce the search space of feasible solutions (which in this case, are all permutations of the N classes) and find the optimal sequence. The restricted DDAG design, presented in section 3 is used as a bounding condition in the algorithm.

Algorithm

1. Initialize current_best = 0.5; Nodes = {(1,N)}
2. Select the topmost node N_t from Nodes
3. Expand the node to its two children, compute the upper bound on the accuracy, and estimate the possible accuracy at this node using the restricted uniform probability case. Use $q = \max q_i, i \in \phi$
4. If the upper bound on the accuracy at the current node is greater than the current_best, explore the node by going to step 3, else skip the node and start from next node in Nodes.
5. If the node is a leaf node, then compute the accuracy, if it is better than the existing current_best then make the current accuracy as current_best
6. Continue this until the list Nodes is empty

This algorithm has been tried out with minimal success, as the bound for branching is very weak, and almost all the times results in exhaustive search.

Complexity of the DAG Design Problem

A DDAG design problem is much more complex than the Optimal Binary Decision Tree problem, which is an NP Hard problem. Here we show a basic proof of its relation to the Travelling Salesman Problem (TSP), and its complexity related to a TSP.

A list of N classes $\omega = \{\omega_1, \omega_2, \cdots, \omega_N\}$ is given. q_{ij} stands for the misclassification probability between classes ω_i, ω_j. P_i stands for the apriori probability of the class ω_i. The problem is to find the best permutation of the classes, such that the following equation is maximized.

$$J(i,j) = [(1-q_{ij})\rho_{ij}^i + \frac{1}{2}(1-\rho_{ij})]J(i,\psi_r) + [(1-q_{ij})\rho_{ij}^j + \frac{1}{2}(1-\rho_{ij})]J(\psi_l,j) \qquad (4.4)$$

where

$$\rho_{ij}^{i} = \frac{P_i}{\sum_{k=i}^{j} P_k}$$

and

$$\rho_{ij} = \rho_{ij}^{i} + \rho_{ij}^{j}$$

Using this formulation, the objective is to find the class order which maximizes this expression.

Corresponding Decision Problem The corresponding decision problem of a DAG may be posed as whether a DAG exists with a classification accuracy of at least k. The DDAG's accuracy be verified in polynomial time and a decision can be made whether it is greater than k or not.

Reducing TSP to the DAG Design Problem A TSP problem is reduced to a very special case of the DAG design problem. Let P_is be same for all the classes. i.e each class has an apriori probability of $\frac{1}{N}$. Now, equation(4.4) can be modified by introducing a term n, where n is level of tree from bottom which you are working at. When working at the highest level, then all the n classes are under consideration. When working at deciding the single classifier level, $n = 2$, which is the least possible value it can take. In such a case, $\rho_{ij} = \frac{2}{n}$, $\rho_{ij}^{i} = \frac{1}{n}$ and $\rho_{ij}^{j} = \frac{1}{n}$

$$J(i,j,n) = [\frac{(1-q_{ij})}{n} + \frac{(n-2)}{2n}]J(i,\psi_r,n-1) + [\frac{(1-q_{ij})}{n} + \frac{(n-2)}{2n}]J(\psi_l,j,n-1) \quad (4.5)$$

$$= \frac{n-2q_{ij}}{2n}[J(i,\psi_r,n-1) + J(\psi_l,j,n-1)] \quad (4.6)$$

Assume that the left corner is fixed in the beginning itself. Let class a be the first class fixed on the left side.

Now, the left sub problem is nothing but the previous objective function value. Only the right part changes depending upon the addition of new classes. $J(a,\psi_r,n-1)$ stands for the value of the objective function computed till n-1 steps. $J(\psi_l,j,n-1)$ stands for the additional term that gets added at every selection that is made. The traveling sales man problem has the following expression.

$$g(i,S) = \min_{j \in S}\{c_{ij} + g(j,S-\{j\})\}$$

$$J(a,S,n) = \min_{j \in S} J(a,j,n) = \min_{j \in S} \frac{n-2q_{ij}}{2n}[J(\psi_l,j,n-1) + J(i,\psi_r,n-1)]$$

The two equations are similar, in fact the second equation is more complex than the first equation, in the sense that it has a multiplication factor which depends on the current choice we are making, and the second term is a summation of two factors, of which one is

the value of the function obtained till now, added to the cost of selecting a node j from S. This way the equation looks like a more general form of the traveling salesman problem with a multiplication factor, and the additional cost not being a constant and depending on the choice we are making. Thus, DAG Design problem is NP-Hard.

Simulation of Results

Algorithm $DAG2$ finds an arrangement of classes which maximizes J using a depth first branch and bound technique. The objective is to find a list of size N, which is the optimal class order. The solution starts with an empty set S. At each step in the recursion, a new class is added to the left and right branches of the tree, at its ends. *pos* is the variable storing the current vacant position from the center, on both sides, where a class is to be added.

J is a decreasing function of number of classes. As classes are added to the intermediate solution at each step, the classification accuracy of the new classifier is less than the previous one. S represents the intermediate solution. If the classification probability of the intermediate solution is less than the best value of J observed till then, the branch is discarded. This is continued over all possible arrangements, and best one is chosen out of them.

4.3.3 Limitation of the DAG

A DDAG is an efficient way to combine classifiers, however, it suffers from the following problems:

- A DDAG just outputs the class labels, and there is no way to get a second option, or a ranked output of classes.
- There is no way that a class can be rejected, or any other information regarding the classification be provided
- The objective function stated in Equation (4.4), assumes that all the samples reach all the nodes when estimating its required parameters, which is not true.

4.4 Improved DDAG Algorithm

To address the above problems, an algorithm to classify the samples using a DDAG is proposed. The algorithm takes advantage of irrelevance of few classifiers on the path, to obtain a second decision. This information is used in designing the DDAG. The algorithm is described below:

1. Present a sample to the root node of the DDAG

2. Depending on the decision of the classifier, choose the left or right sub-DAG
3. Follow steps 1 and 2 until a leaf node is reached, where the sample is assigned a label p, called the primary labeling.
4. Present the sample again to the root node of the DDAG
5. This time, take all the opposite directions to the ones suggested by each classifier, until a relevant classifier is reached. This relevant classifier can be known using the true label of the class, at the training time.
6. Once a relevant classifier is reached, follow the right path to reach the leaf node. Here the sample is assigned a label q the secondary labeling.
7. If p and q are the same and are equal to the true label t_i of the sample, return the label of the class as p and increment the number of correct samples by 1
8. If p and q are same and are different from the true label, return the label of the class, and increment an error.
9. If p and q are unequal, then the decision is not confident, and reject the sample, which can be later handled using other approaches like majority voting.

4.4.1 Greedy Approximate Algorithms

Maximizing the accuracy of the DDAG using the objective function stated in Eqn (4.3) is a permutation selection problem, and is an NP-complete problem. Therefore, there is no polynomial time algorithm currently existing for this. Using the properties of a DDAG, few greedy approximate algorithms to maximize to the objective function are presented in this section.

The following characteristics of a DDAG guides the choice of the algorithms:

1. In a DDAG, the classes placed at the end of the list have the highest relevant path, and are the most error prone classifications.
2. If the nodes in the top few levels of the DDAG are good, the critical classifications can be done more robustly. By measuring the performance of the classifier in terms of margin, or number of support vectors, or accuracy on the validation dataset, the pairs with good classifiers can be moved on to the edges.

This algorithm discusses the design of DDAG using the apriori probabilities. The idea is to put the high probable classes in the center of the list, and the low probable classes to the edges of the list.

The above algorithm shows improved results only when the misclassification probabilities of the classes are almost the same. However, if that is not the case, and the priors are

Algorithm 2 PriorSort

Input: Classes ω_i, Apriori ProbabilitiesP_i
Output: Outputs a DDAG with maximum classification probability as an ordered tuple S

$S = ()$
while ω is not empty **do**
  Select two classes $\omega_i, \omega_j$with highest aprioris
  $S = (\omega_i, S, \omega_j)$
  $\omega = \omega - \{\omega_i, \omega_j\}$
end while
output S

approximately equal, as in the case of many datasets (like vowel, letter, etc), the above algorithm provides no improvement to the DDAG. In these cases, using a margin based approach, where large margin classifiers are placed at the top of the DDAG and the small margin classifiers are introduced later into the DDAG. This ensures that the classifiers for classes which are placed at the ends has very good classification probability, measured in terms of higher margins.

Algorithm 3 MarginSort

Input: Classes ω_i, Margins M_{ij}
Output: Outputs a DDAG with maximum classification probability as an ordered tuple S

$S = ()$
Compute the average margin for a given class with all the other classes.
while ω is not empty **do**
  Select the class with highest average margin from the set of classes ω, and pick the pairwise
  classifier with highest margin M_{ij} with that class, and introduce it at the edges.
  Insert the pairwise classifier at the top of the DDAG.
  $\omega = \omega - \{\omega_i, \omega_j\}$
end while
output S

However, when both priors and misclassification's are available and vary a lot, there is no preference to either of them given. Both of them have to be considered while designing the DDAG. In this algorithm, we make use of the Q_i defined as the overall misclassification probability of class i, computed as the average error it makes with all the other classifiers.

Large priors push the classes to the center of the list and also the large misclassifications. Therefore, using a product of these two gives an estimate of the error that the class can expect. Using this, the high probable and high error classes are put in the center and the rest are placed towards the edges. The algorithm for doing this is described below. The value of Q_i is computed as

$$Q_i = \sum_{j=1}^{N} s_{ij}^{ij}.$$

The overall weight of a class is computed as $W_i = P_i Q_i$.

Algorithm 4 PriorAndMisclassification

Input: Classes ω_i, Split Probabilities $\mathcal{S} = s_{ij}^{k}$
Output: Outputs a DDAG with maximum classification probability as an ordered tuple S

$S = ()$
for i = 1:N **do**
  Compute W_i.
end for
while ω is not empty **do**
  Select the class with highest weight W_i
  Insert it into the closest vacant position from the center
  $\omega = \omega - \{\omega_i\}$
end while
output S

4.4.2 Simulation Results of the Various Classifiers

Table 4.2 shows results of simulation on the various UCI [40] datasets. The priors of the classes from each dataset are computed. Pairwise classifiers are trained on each dataset, and the margins obtained at each pairwise classifier is stored. All the samples of the training dataset are presented to each pairwise classifier to obtain the split probabilities S_{ij}^{ki}. These are used to compute the possible accuracies in various cases using the objective function as defined in Eqn 4.4.

In Table 4.2, it can be seen that there is a considerable difference between the best and worst performances possible by a redesigned DDAG. In most cases, the greedy algorithms have performed close to the average accuracy of the DDAG obtained by redesigning. In cases where there is a large variation in priors of the dataset, as in the case of Satimage, the prior sort algorithm has performed the best. In the case where there is a minimal variation in the priors as in the case of letters, the algorithm considering prior and misclassification

Dataset	Best	Average	Worst	PriorSort	MarginSort	Prior-Misclass
Satimage	99.25	94.40	86.17	99.19	93.16	87.48
Glass	97.59	84.93	70.53	71.98	75.50	70.54
Letter	98.64	96.71	95.67	96.07	96.02	97.28
Pendigits	99.84	98.86	97.50	99.44	98.74	97.52
Optdigits	99.97	99.75	99.45	99.90	99.80	99.88

Table 4.2: The best, average, worst case probabilities of misclassifications for various permutations of a DDAG arrangement

together has performed the best. This shows that depending on the variance observed in priors and misclassifications, the best performing greedy algorithm can be chosen.

To demonstrate the dependence of the improvement and accuracy on the variance of the priors and split probabilities, two simulation experiments are conducted on synthetic datasets. The first experiment fixes the prior of the classes, and varies the misclassification probabilities. A large number of random priors and misclassifications are generated, and the average of the accuracy computations is presented in Figure 4.4. The improvement obtained as the variance in split probabilities increases is shown in Figure 4.5. The second experiment fixes the split probabilities, and varies the priors. The different accuracies obtained by increasing variance in the priors is shown in Figure 4.6, and the improvement possible is plotted in Figure 4.7.

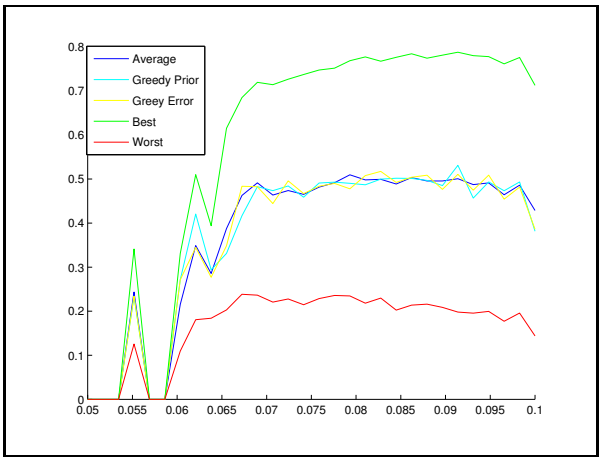

Figure 4.4: Varying split probabilities (S), constant priors (P), showing the best, worst, greedy with priors alone, with priors and misclassifications

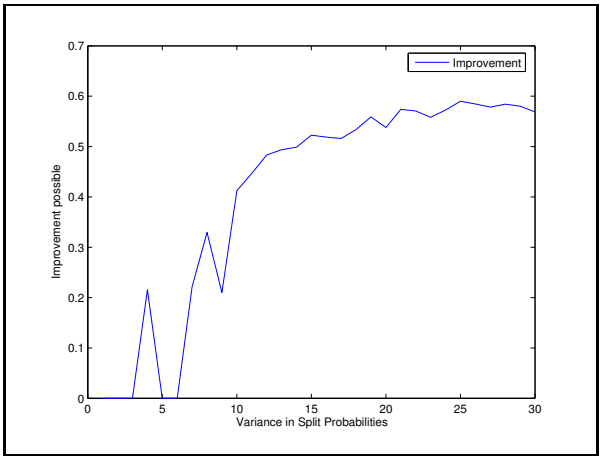

Figure 4.5: Varying S, constant P, computed as best minus worst

4.5 Summary

The performance of an ensemble of pairwise classifiers to emulate multiclass classification system depends on parameters from data, individual pairwise classifiers, and the architecture used. A novel formulation for the design of multiclass classifier is presented based on this idea. Algorithms for designing better multiclass classifiers using the DDAG architecture are proposed. The algorithms proposed are approximate, as the problem of building optimal DDAG is a permutation selection problem, and is difficult to solve in polynomial time. Future work involves improving the approximate algorithms such that they are closer to the optimal DDAG.

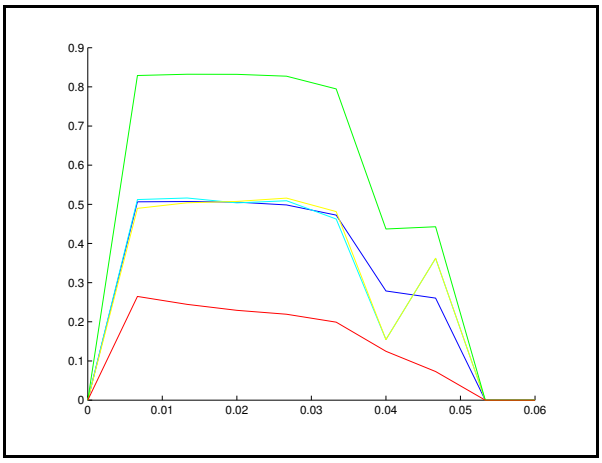

Figure 4.6: Varying P and constant S, showing the best, average and worst, greedy with priors alone, with priors and misclassifications

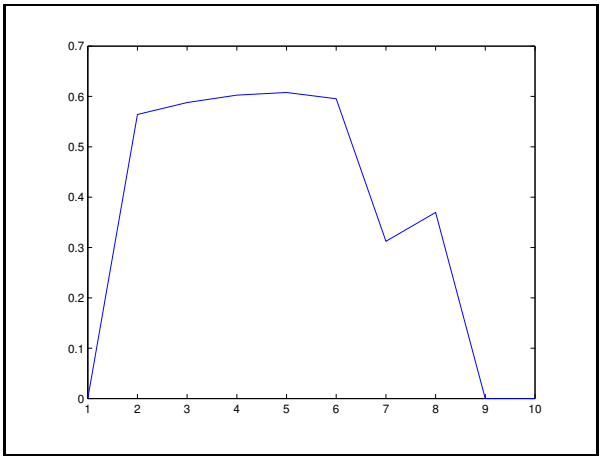

Figure 4.7: Varying P and constant S, Computed as best minus worst.

Chapter 5

Identification of Class Hierarchies

In the previous chapters, the effectiveness of DDAG and improvements possible are shown. In spite of its robustness, a DDAG suffers from the problem of large size and testing time complexity as it employs a large number of classifiers. In this aspect, a Binary Hierarchical Classifier (BHC) is an efficient representation of the classification knowledge in the form of a tree. This requires much lesser component classifiers than the DDAG. In this section, a closer look on the tree based classifiers is taken, and algorithms are presented to design better hierarchical classifiers.

Many practical pattern recognition applications such as character recognition, biometrics etc, have multiple classes of patterns to recognize. Binary classification is a well researched area with exhaustive theoretical analysis. Many of these can be extended for multiclass recognition problems. Few direct extensions of binary classifiers exist for multiclass classification. However, it is usually preferred to solve the multiclass classification using a combination of binary classifiers [16, 29]. Such combinations are found to outperform the direct multiclassifiers empirically [41] and theoretically [67].

For an N class classification problem, one-vs-rest approaches build N classifiers, each with samples from one class as positive, and the rest as negative. The decision of the most confident positive result of a classifier is assigned to the sample. The distributions of the positive and negative samples in this case are not compact and bounded. Therefore the classification boundary needs to be highly complex, which results in poor generalization. These approaches are shown to be inferior to one-vs-one approaches in both accuracy, as well as training times [82]. One-vs-one methods train $\frac{n\cdot(n-1)}{2}$ classifiers, one for each possible pair of classes. These are usually combined using voting approaches. The Decision Directed Acyclic Graphs(DDAG) provides a very efficient way of combining them hierarchically [67]. These combined classification systems usually have high performance in terms of accuracy. However they are large in size and take longer times for classification.

It is shown that hierarchical classifiers are both efficient [82] and accurate [54, 41] for most pattern classification tasks. Design of hierarchical classifiers is still a relatively unexplored

area, except for the decision trees. Large amount of literature exists for the design of the decision trees [68, 61, 73]. They employ heuristics to identify the features (say based on an information theoretic measures) best suited for the classification at every stage of the classification hierarchy. Many pattern classification problems in character recognition systems, data mining etc. employ decision trees. Automatic hierarchical classification has also been effectively used in hyperspectral data analysis [54], text classification [59] etc.

We focus on building hierarchy of classes, rather than features, for designing efficient and accurate classifiers. Deviating from the decision tree paradigm, we depend on a class-based hierarchy for the classifier design. Building class-based hierarchy is superior to that of feature-based hierarchy because of the following reasons:

- Class-based division is more intuitive than feature based division as natural groups of classes exists for many problems. It can be observed evidently in applications like text categorization and character recognition.
- Class-based division results in a predictable number of component classifiers unlike feature-based hierarchies.
- When large number of classes exists when compared to features (eg. OCR problems), feature based partitions are insufficient to express the complete classifier.
- Class-based partitioning schemes employ the natural labeling of the samples for grouping. While feature-based grouping needs not result in directly useful partitioning at every stage.

In most problems, one will not be able to induce to structure on the distribution of classes. They can be quite arbitrary and problem dependent. This makes the building hierarchies of classes not amenable to the conventional clustering schemes. We propose to build a graph to describe the distribution of classes (in Section 5.3) and formulate the classification problem as successive graph-partitioning problem (Section 5.4). We argue that none of the popular objective functions can directly solve the problem for any arbitrary data set. We propose an integrated scheme which picks the best available partition (Section 5.6) for building class-hierarchies. We then argue that the partitioning problem algorithm need not yield mutually exclusive sets of vertices. In fact, the overlapping partitions make the classification problem less complex at every stage of the hierarchy. Complexity of the classification problem is measured in terms of the number of support vectors employed (Section 5.5). Finally we demonstrate that the proposed algorithm yields excellent results in Section 5.7.

5.1 Problem Statement

We formulate the problem of building hierarchies in a graph-theoretic framework. Let $\Omega = \{\omega_1, \ldots \omega_N\}$ be the N classes and $\mathcal{G}$ be a fully connected graph corresponding to the

distribution of the classes. Vertices of the graph $\mathcal{G}$, $v_1, \ldots v_N$ correspond to the classes; and edges E_{ij} relate to the relative misclassification of the classes ω_i and ω_j. Let $\mathcal{G} = (V, E)$ be a weighted undirected graph, where V is the set of $|V| = N$ nodes, and E is the set of $|E| = \binom{N}{2}$ edges. A cut is a set of edges, removal of which results in partitioning of the vertices into two mutually exclusive sub sets of vertices. Let $C = C_1, \cdots, C_k$ be the subgraphs resulting from the cuts Each C_i is called as a cluster or a partition, which can be represented as a sub-graph of G. The edges found in any of the partitions form the intra-cluster edges, and the edges which are cut to obtain the sub-graphs form the set of inter cluster edges.

A linear cut LC is a cut such that there exists a hyperplane which will pass through the set of edges corresponding to the cut $\mathcal{C}$.

The problem we address here is to find successively a sequence of linear cuts such that all the individual nodes in the graph gets isolated. It can be easily shown that the minimum number of cuts needed for this is $\log N$ and the maximum number of cuts one may need is $N - 1$. There exists two hierarchical classifiers corresponding to these best and worst cases – BHCs and DDAG. The sequence of cuts can also be understood as a sequence of binary hierarchical classification where at each stage, the cut defines two new graphs and the nodes at the next level correspond to the cuts on the new graphs.

5.2 Related Work

Hierarchical classification is popularly achieved by building hierarchy over features or classes. Some background on the feature based hierarchies is provided, followed by the more promising approach of class based hierarchies.

5.2.1 Feature Based Hierarchies

These are the class of hierarchical classifiers in which each decision making node is a function of a subset of features used to represent the sample. These functions are either specified manually or can be learnt from the data as in the case of decision trees.

The classifier architectures based on manual hierarchies correspond to a hierarchical representation of a set of rules, which function as an expert system [58, 5, 12]. They encode a set of rules given by a single expert, based on his knowledge of the problem. These systems cannot improve themselves with the help of more labeled examples. A high correlation had to be maintained between the perceptual similarity of the objects and their feature representation in order to obtain hierarchies with good performance.

Instead of manually building the rules based on features, learning them is always preferable. This makes the algorithms independent of feature representations, and uses the training data to come up with a hierarchy. Decision trees [68] are classical approaches to attempt a hierarchical representation of the classification knowledge. They were popular for

organizing data and efficient for classification as the number of comparisons required is of logarithmic order of the number of classifiers.

5.2.2 Class Based Hierarchies

These are class of hierarchical classifiers where the data set is partitioned into multiple subsets at each decision making node depending on the classes of samples. Number of nodes in these classifiers are usually of the linear order of the number of classes, which is often small when compared to feature based hierarchies.

Decision Directed Acyclic Graph is an efficient way to combine the pairwise classifiers, to overcome the quadratic evaluation time of the pairwise majority classifier [67]. This connects the pairwise classifiers as a rooted binary directed acyclic graph. The sample is initially presented to the root node of the graph, and depending on the decision taken, either left or right sub-DAG is selected for next evaluation, until a leaf node is reached. The sample is assigned the label of the leaf node it reaches. The classification time is significantly brought down to $O(N)$ from $O(N^2)$ of pairwise majority. However, the size of the classifier still remains large. The architecture is fixed for a given set of classes irrespective of the training data and its feature representations.

Automatic learning of class hierarchies, resulting in a binary tree architecture, is a powerful approach for multiclass classification. The size of these classifiers is of $O(N)$ and the testing time is $O(\log N)$. Binary Hierarchical Classifier (BHC) [54, 13] is one such method, which learns the best partitioning of the classes into two subsets at each node of the tree using a deterministic annealing algorithm. It builds $N-1$ classification nodes for an N class problem. DB2-SVM [82] attempts to build a hierarchical SVM using sample level clustering at each node. A clustering algorithm divides the classes in the data into two subsets, and an SVM is built. Although these algorithms are extremely efficient in time and space, they sometimes compromise on accuracy. Each node in this architecture is decided depending on the training data. The tree is homogeneous in nature, which requires the same classifier to be used at all the nodes irrespective of the generalization issues.

5.2.3 Graph-based formulation of partitioning problem

Partitioning the set of samples/classes into two subsets can be addressed using any clustering algorithm [82, 54, 13]. However, the distribution of the classes (not the samples within the classes) may not be suitable for these clustering algorithms. Typical assumptions like compactness of the classes and Normal distribution are violated on the set of classes. Therefore we build a graph of the classes (vertices being the classes and edges being the similarity/separability of them) and employ graph-based partitioning schemes for building hierarchy of classes. This class of partitioning scheme does not impose any structural assumptions on the data to be clustered.

For a hierarchical architecture, graph-based partitioning is a natural choice. An ideal condition to split the classes into subsets is that the closest pair from the two subsets will be the farthest among all the partitions. More over, one should be able to design a classifier which can separate these classes. The problem of partitioning classes into optimal subsets may not have a polynomial time solution. However, there are many approximate algorithms which can provide solutions identical to the one interested in many popular situations

5.3 Graph Formulation

Let w_{ij} represent the weight of the edge E_{ij} connecting the vertices V_i and V_j. Define $w(C_i, C_j)$ as the sum of weights of edge weights between nodes in C_i and C_J,

$$w(C_i, C_j) = \sum_{V_i \in C_i, V_j in C_j} w_{ij}$$

This is called as a *cut*. The partitioning of a graph is usually posed as finding the *cut* such that the cost of the cut is maximum or minimum, with or without constraints.

The desired characteristics of the resulting partitions, expressed as an objective function, such as the sum of all intra cluster edge weights is minimum for a compact clustering, or the sum of all inter-cluster edge weights must be maximum for the maximum separated clusters. The edge weights can be metric distances like Minkowski, or non-metric distances like Tanimoto distance, or similarity measures like the cosine distance or even the correlation distance. Each vertex of the graph can be the samples from a class or any representative point of the classes(eg. mean).

As mentioned above the vertices of the graph could be

- the individual training sample or
- the class representative.

Accordingly, the definition of the edge weight will vary. Popular useful measures which can be used for graph-formation include:

5.3.1 Edge Weights for Points as Vertices

The edge weights of the graph are defined similarity or dissimilarity of the vertices. A few broad classes of measures which can be used as distance measures for clustering is given below.

Minkowski Metric Minkowski Metric measures the separation between the samples in the space and is the most frequently used distance definition. Minkowski distance between

two points x and x' is defined as,

$$d(x, x') = \left(|x_k - x'_k|^q \right)^{1/q}.$$

This is a generalized distance measure which on setting $q = 1$ results in city block metric, and $q = 2$ in the popular Euclidean metric.

Cosine Distance This measures the cosine of the angle subtended by the line joining the points at the origin. The farther the points, the larger the angle subtended and smaller the cosine. This is hence a similarity measure.

$$s(x, x') = \frac{x^t x'}{|x||x'|}$$

Tanimoto Measure This is the ratio of number of shared attributes to the total number of attributes (the ratio of cardinality of the intersection to the cardinality of union) .

$$s(x, x') = \frac{x^t x'}{d}$$

This is a distance useful in the case of clustering samples with binary features. A variant of this is to compare the number of shared attributes with those in either x or x'.

$$s(x, x') = \frac{x^t x'}{x^t x + x'^t x' - x^t x'}$$

5.3.2 Edge weights for Sets as Vertices

The vertices in the graph can be represented as sets of points, each set belonging to a particular class. Distance measures employed for characterizing their separability is described below.

Average KL - Divergence This is the distance between the probability distributions p and q of the classes ω_i and ω_j, measured as

$$d_{kl}(\omega_i, \omega_j) = \sum_{i=1}^{N} (p(i) - q(i))(log(p(i)) - log(q(i)))$$

Minimum Distance This is equal to the distance between closest pair of points from different clusters.

$$d_{min}(D_i, D_j) = \min_{x \in D_i, x' \in D_j} |x - x'|$$

Maximum Distance This is measured as the distance between the farthest pair of points from different clusters.

$$d_{max}(D_i, D_j) = \max_{x \in D_i, x' \in D_j} |x - x'|$$

Average Distance The average of all pairwise distances, where the pairs are formed using points from different clusters.

$$d_{avg}(D_i, D_j) = \frac{1}{n_i n_j} \sum_{x \in D_i} \sum_{x' \in D_j} |x - x'|$$

Mean Distance The distance between the means of the clusters.

$$d_{mean}(D_i, D_j) = |m_i - m_j|$$

5.4 Objective Functions and Partitioning Strategies

Partitioning is a key step to build hierarchical classifiers. The partitions which satisfy different objectives can be obtained by characterizing the requirements of the classification task. In building a hierarchy, a partitioning problem with a different set of classes is encountered at each node, and requires a suitable partitioning strategy. Few popular objective functions used for graph-based partitioning are described below.

5.4.1 Objective Functions

Ratio Association This objective function measures the within cluster association, relative to the size of the cluster. This is defined as

$$RA(C) = \max_{C_i} \sum_{i=1}^{k} \frac{w(C_i, C_i)}{n(C_i)}$$

Let n_i be the number of samples in C_i, and let m_i be the mean of those samples,

$$m_i = \frac{1}{n_i} \sum_{x \in D_i} x.$$

The sum of squared errors is defined by

$$J_e = \sum_{i=1}^{c} \sum_{x \in D_i} |x - m_i|^2 = \sum_{i=1}^{c} \sum_{x, x' \in D_i} \frac{|x - x'|^2}{|D_i|}.$$

This is the most popular squared error minimization problem, which is same as the ratio association.

Ratio Cut The ratio cut attempts to minimize the cut between the clusters and the remaining vertices.

$$RA(C) = \min_{C_i} \sum_{i=1}^{k} \frac{w(C_i, C \backslash C_i)}{n(C_i)}$$

If the distance measure is used instead of similarity measure, this represents the cut between the partitions maximizing which is the popular max-cut algorithm.

Normalized Cut The normalized cut is a popular partitioning objective, which minimizes the cut relative to the degree of a cluster, instead of its size.

$$NCut(C) = \min_{C_i} \sum_{i=1}^{k} \frac{w(C_i, C) \backslash C_i}{degree(C_i)}$$

Several graph based algorithms exist for partitioning the graph, of which minimum spanning tree algorithms and cut based algorithms are popular.

Algorithms for obtaining minimum cuts prefer isolated classes in the data set to be cut. This is not of much use, as the classification problem between the rest of the classes still remain difficult and large size. In such cases, normalizing the cuts prefers equal size clusters resulting in classifiers of smaller depth. The isolated classes can be separated easily at a later stage.

5.4.2 Partitioning Strategies

Graph Cut based Partitioning

A graph-cut partitions the vertices of a graph into two disjoint subsets. A usual objective is to maximize the value of the cut between the two disjoint subsets. The cost of a cut is defined as the sum of the weights of the edges connecting the vertices, each one selected from a different subset. Finding a cut of optimal cost is NP-hard [77]. Although there are an exponential number of such partitions, finding the minimum cut of a graph is a well-studied problem and there exist efficient algorithms for solving it. This problem can be efficiently solved by recursively finding the minimum cuts that bisect the existing segments as proposed in [86]. There are many ways in which the above optimization functions can be maximized or minimized accordingly. The usual approach is to use greedy algorithms for obtaining an approximate cut. These greedy algorithms are commonly used for partition based clustering algorithms like K-means. For many criterion functions, these greedy algorithms have been shown to converge to a local minima [48].

In this work, we use a clustering toolkit called CLUTO [48]. CLUTO uses a greedy optimization with two phases – initial clustering and cluster refinement. In the initial clustering phase, a clustering solution is computed using a simple nearest cluster assignment, by choosing few random samples as initial clusters. Later, these clusters are refined iteratively.

Hierarchical Agglomerative Clustering

Unlike the partitioning algorithms that build the hierarchical solution for top to bottom, Hierarchical agglomerative approaches are bottom up approaches, which start with n singleton clusters, and form a hierarchy of clusters by successively merging the closes clusters. The key parameter in agglomerative algorithms is the method used to determine the pair of clusters to be merged at each step. In most agglomerative algorithms, this is achieved by selecting the most similar pair of clusters, and there are usually many ways to compute the similarity between two clusters. In our approach, we tried two hierarchical agglomerative algorithms called signal linkage and complete linkage. When $d_{min}(.,.)$ is used as a distance measure between the clusters, the algorithm is called the nearest-neighbor clustering algorithm. If there is a path from one point to other, such that each individual weight of the edges is smaller than a threshold, then the points belong to the same cluster. This clustering thus prefers long clusters. When $d_{max}(.,.)$ is used as a distance measure between the clusters, the algorithm merges those clusters such that the maximum pairwise separation between points in two clusters would be lesser than a threshold. Hence, the algorithm discourages the growth of elongated clusters and prefers compact clusters. Every cluster forms a complete subgraph of the graph representing the dataset.

Other Methods

Other popular methods for graph partitioning include repeated bisection which is a multiway graph partitioning algorithm. This uses a set of repeated greedy graph bisections to obtain an approximate clustering [92]. An alternate way is to use more powerful optimizers such as those based on the spectral properties of the similarity matrix of the samples or various multilevel optimization methods [50, 49].

5.4.3 No Free Lunch Theorem and Superiority of No Algorithms

Class distributions in any pattern classification task are modeled with the help of appropriate distributions. However the relative configuration of these classes are quite random to model or predict. In some of the multiclass classification problems like OCRs, there could be similar classes due to the presence of a problem structure. However, such an assumption is not valid for a general classification problem.

We tested many partitioning schemes on diverse data sets, results of which is depicted in Table 5.1. It can be seen that no algorithm is superior to other algorithms for the hierarchical classification task with the help of binary partitions. In all these cases, we have $N-1$ nodes in the classification tree with varying amounts of performance. We have observed that the performance of these nodes vary across the algorithms due to the difference in margin it obtains and the difference in the structure/topology of the tree.

	I1	K-means	Maxcut	NCut	Clink	Slink
yeast	53.5	53.5	50.13	50.13	53.5	48.11
glass	51.85	51.85	55.56	55.56	51.85	59.26
satimage	81.7	81.7	81.45	81.45	81.7	81.7
pendigits	88.36	88.36	83.45	83.45	86.54	86.51
optdigits	94.32	94.32	93.93	93.93	94.16	93.21
letter	68.6	68.6	62.9	62.9	63.25	65.5

Table 5.1: Results of various partitioning Schemes for Building Hierarchies

5.4.4 Integrated Partitioning Scheme

For a given dataset, it is impossible to predetermine what clustering algorithm results in a cut that gives best classification performance. Moreover, in a hierarchical framework, where there are different classification problems at each level of the hierarchy, it is even more difficult to fix a global clustering algorithm, which performs better at all the nodes for all the datasets. Each clustering algorithm utilizes different properties of the dataset to cluster them together. Exploiting the advantages of different partitioning schemes for improving the performance, we propose an integrated partitioning scheme which selects the best algorithm at every stage of the design in a greedy manner. The procedure to build the complete tree is presented as the routine *BuildTree* in Algorithm (5). This recursively builds the tree using the routine *GetBestPartition*, shown in Algorithm (6) which runs different partitioning algorithms on the given set of classes, and returns the best partition.

Algorithm 5 BuildTree(Ω)

Input The algorithm takes in a set of classes Ω

Uses Partition is a binary vector storing the label of cluster for each class. The function *Split* applies this binary vector to Ω and results in two subsets of classes Ω_l, Ω_r.

Output Builds a complete tree, with greedily best partition at each node.

partition = GetBestPartition(Ω);
(Ω_l, Ω_r) = Split(Ω,*partition*);
if $n(\Omega_l) > 1$) **then**
  BuildTree(Ω_l);
end if
if $n(\Omega_r) > 1$) **then**
  BuildTree(Ω_l);
end if

Algorithm 6 GetBestPartition(Ω)

Input Set of classes $\Omega = \{\omega_i, i = \cdots N\}$, List of partitioning methods to be used $\mathcal{M} = \{m_i, i = 1 \cdots K\}$

Output Returns the partition with maximum margin

for each clustering algorithm $m_i \in \mathcal{M}$ **do**
  $partition_i \leftarrow$ Partition(Ω)
  $margin_i \leftarrow$ TrainSVM($partition_i$)
  if $best\_margin < margin_i$ **then**
    $best\_margin = margin_i$
    $best\_partition \leftarrow partition_i$
  end if
end for
return *best_partition*

Dataset	Accuracy	NNodes	Min Margin	Avg Margin	NSV	AvgNSV	Sum Margin
glass	55.5556	5	0.1141	0.2215	237	47.40	1.11
yeast	50.9434	9	0.0429	0.1433	2024	224.89	1.29
satimage	81.6500	5	17.6048	22.3284	3131	626.20	111.64
pendigits	87.1641	9	8.8218	13.3955	3592	399.11	120.56
optdigits	93.6004	9	3.4863	4.9021	1681	186.78	44.12
letter	63.6500	25	0.0352	0.0589	27119	1084.76	1.47
malayalam	99.6610	115	1.1033	2.6639	4148	36.07	306.35

Table 5.2: The performance of the proposed algorithm using non overlapping partitions at each node

Performance

The new integrated scheme resulted in classification results as shown in Table 5.2

In most of the cases, we have observed that the accuracies are improved with the integrated partitioning scheme. However, due to the sub-optimal selection of the partition hierarchically the algorithm is unable to produce the best hierarchy in all the cases. An improvement in the average margin, and a reduction in the average number of support vectors per node are however observed in all the cases.

5.5 Margins, Support Vectors and Acceptability of a Partition

Support vector machines are classifiers learnt with the objective of maximizing the margin of separation between two classes in a dataset. The maximization of margin is equivalent to minimizing the norm of the weight vector [44]. The problem of learning an SVM is thus, to minimize the $|w|^2$ such that the data samples are classified correctly. In a linearly non separable case, a penalty term is added for each misclassification in the form of slack variables [44].

After learning, each support vector x_i has a coefficient α_i, which specifies its contribution to the decision boundary. The value of the margin γ obtained can be expressed as

$$\gamma = \frac{2}{|w|} = \frac{1}{\sum_{i=1}^{N} \alpha}$$

And the expected error $\mathcal{E}_N[error]$ of this support vector machine can be measured in terms of number of support vectors in the dataset, and can be expressed as

$$\mathcal{E}_N[error] \leq \frac{\mathcal{E}_N[N_s]}{N} \tag{5.1}$$

It is desirable to obtain a large value of margin and to express the decision boundary in as minimal number of support vectors as possible to achieve a good generalization.

The number of support vectors thus provide an effective measure for measuring the usefulness of the partition.

5.6 Non-overlapping Partitioning Scheme

The proposed partitioning scheme aims at improving the margin at each partition obtained by different clustering schemes, and choosing the best partition out of it. Initially, a clustering algorithm is used to partition the dataset into two clusters. On this dataset, an SVM with very low penalty on misclassifications is trained. This biases the SVM to maximize the margin even at the cost of misclassifications, which results in a large number of support vectors. The probability of error of a support vector machine depends on the number of support vectors. If the probability of error is large, the support vector machine performs bad on the class. The probability of error of support vector machine on a class ω_i can be estimated using the number of using the equation 5.1. If we define a tolerance level θ such that, if the error of the global SVM on this class is more than θ, remove the class from training. This achieves a high margin at the node of the tree. These removed classes are added into both the children of the node. At each node, the process repeats recursively, resulting in overlapping partitions.

When the above clustering methods are applied, and an improvement in margin is obtained, we need to pick the partitioning with maximum improved-margin. The algorithm can be summarized as

- For method i, say we obtain a partition i, Method(i) $\rightarrow$ Partition(i)
- Each partition results in an SVM margin, Partition(i) $\rightarrow$ margin (i)
- The SVM margin can be improved by removing the bad classes out. Margin(i) $\rightarrow$ bettermargin(i). The bad classes can be removed by counting the percentage of support vectors contributed by each class to the complete set of support vectors.
- Pick that partition out of all the partitions, that has the highest improved margins

The algorithm is formally described in Algorithm 7.

Algorithm 7 BuildOverlapTree(Ω)

Input The algorithm takes in a set of classes Ω
Uses Partition is a binary vector storing the label of cluster for each class. The function *Split* applies this binary vector to Ω and results in two subsets of classes Ω_l, Ω_r.
Output Builds a complete tree, with greedily best partition at each node, with overlapping partitions.

partition = GetBestPartition(Ω);
(Ω_l, Ω_r) = Split(Ω,*partition*);
(Ω_{int}) = RemoveBadClasses(θ);
if $n(\Omega_l \cup \Omega_{int}) > 1$) **then**
  BuildOverlapTree(Ω_l);
end if
if $n(\Omega_r \cup \Omega_{int}) > 1$) **then**
  BuildOverlapTree(Ω_l);
end if

Proposition 1 The number of nodes in the binary hierarchical tree built over n classes with an overlap of utmost k classes at each node is

$$f(n,k) = 2^k(n-k) - 1$$

Proof The proof is shown in two steps, first by counting the number of nodes in the trees obtained by allowing an overlap of exactly k nodes from the tree, and the second is extending it to an overlap of at most k nodes.

Consider a worst case scenario, where a node with n nodes is always split into two subsets with cardinality of $n-k$ and $k+1$. Since we are allowing an exact overlap of k nodes, no further k overlap splits are possible in the node with $k+1$ elements. Let the number of nodes in the $k+1$ class node be equal to $f(k+1)$. In exact removal case, $f(k+1) = k$, as $k+1-1$ is the number of nodes possible in the sub-tree without overlap. However, we continue to use the $f(k+1)$ for deriving the expression for the "at most k" case.

By expanding the recursion, in the exact k removal case, using $f(k+1) = k$

$$\begin{aligned} f(n,k) &= f(n-1,k) + k + 1 \\ f(n-1,k) &= f(n-2,k) + k + 1 \\ &\vdots \\ f(k+1,k) &= k \end{aligned}$$

Which when added up, results in

$$f(n,k) = (n-k)(k+1) - 1$$

However, in the above derivation, we have considered $f(k+1)$ to be fixed as k, which is actually not. Considering at most k removal, the equation would be

$$\begin{aligned} f(n,k) &= f(n-1,k) + f(k+1,k-1) + 1 \\ f(n-1,k) &= f(n-2,k) + f(k+1,k-1) + 1 \\ &\vdots \\ f(k+1,k) &= f(k+1,k-1) \end{aligned}$$

Summing up,

$$f(n,k) = (n-k)(f(k+1,k-1)+1) - 1$$

Removing the recursion on k directly gives the number of nodes in the tree with k overlaps built over n classes. Writing $f(k+1,k-1)$ as $g(k)$,

$$\begin{aligned} g(k) &= ((k+1)-(k-1))g(k-1) + 1 \\ &= 2g(k-1) + 1 \end{aligned}$$

This expands to a geometric progression with a ratio of 2 between numbers, and it sums up to

$$g(k) = 2^k - 1$$

Substituting this in the exact case we get,

$$\begin{aligned} f(n,k) &= (n-k)(2^k - 1 + 1) - 1 \\ &= 2^k(n-k) + 1 \end{aligned}$$

Therefore, the worst case size of a binary tree built on n classes, with at most k overlap is given by

$$f(n,k) = 2^k(n-k) + 1$$

Proposition 2 The average case depth of a binary tree built on n classes with k overlap is approximately k greater than the non-overlap tree.

Proof From the above proposition, we can say

$$f(n,k) = 2^k(n-k) + 1$$

The average depth of a tree is of the logarithmic order of the number of nodes, on average. Taking the logarithm of $f(n,k)$, it can be easily shown that the depth of the tree does not increase more than k, when at most k classes overlap at each node.

5.7 Results and Discussions

This section presents the experiments conducted to evaluate the above proposed algorithm. The evaluation of a classifier is made in terms of the following parameters

Experimental Methodology The experiments to test the proposed improvements in the hierarchy building algorithm are conducted on the UCI datasets [40]. In all the cases, the training set and testing set are used as is, if they are available. In the absence of a separate test set, the training set is split into 70% for training, and the rest for testing. In each case a Linear SVM is used, and is implemented using the SVMLight [45]. The partitioning at each stage is done using the CLUTO [48] toolkit, using correlation and cosine distances. Graph based and agglomerative clustering algorithms are used from the CLUTO's implementation.

Performance Evaluation

- The accuracy obtained using various clustering algorithms and proposed clustering algorithm, as shown in Table 5.3.
- The size of a tree is an important parameter when considering graph based classifiers. This is measured as the number of nodes in the classifier. This value for each clustering algorithm and the proposed algorithm is presented in Table 5.4.

	I1	K-means	Maxcut	NCut	Clink	Slink	Proposed
yeast	53.91	53.91	53.71	53.71	53.91	54.11	53.70
glass	55.56	55.56	40.74	40.74	55.56	55.56	55.55
satimage	81.25	81.25	81.05	81.05	81.25	81.4	81.95
pendigits	89.11	89.11	90.17	90.17	89.34	90.97	91.88
optdigits	94.77	94.77	94.38	94.38	94.27	94.82	94.93
letter	71.75	71.75	70	70	69.95	70.7	73.45

Table 5.3: Accuracies obtained on the UCI datasets by removing atmost 2 nodes at each node.

- The margin at a node determines the overall performance of the node. In a hierarchical classifier, the weakest link in the whole classifier is the node with least margin. The improvement in minimum margin using the proposed algorithm is demonstrated in Table 5.5.
- The overall performance of the classifier can be characterized using the average margin of the whole classifier. The improvement on the average margin using the proposed approach is shown in Table 5.6.
- The speed and the generalization ability of the classification depends on the number of support vectors per node, and the depth of the tree. The lesser the number of support vectors, the better the generalization and the classification speed. Different partitions can be compared using the number of support vectors, the number of classes removed from each partition, and the expected error bound of a support vectors expressed in terms of number of support vectors N_s and total samples N as,

  $$E[error] \leq \frac{N_s}{N}$$

  The reduction in the average number of support vectors using the proposed method is shown in Table 5.7

5.7.1 Improvement in Margin

At each level of the hierarchy building, an initial SVM is trained with a parameter setting of very less penalty for a misclassification. This gives larger margins resulting in a large number of samples in each class labeled as support vectors. Now, the "bad classes" are selected from this partition. A class is considered "bad" if more than 20% of its samples are support vectors. At each step of partitioning, if it is possible to remove bad classes, a new SVM with these classes removed is trained. As an example, the improvement of margin obtained at few nodes in the proposed hierarchical trees are shown in Table 5.6.

	I1	K-means	Maxcut	NCut	Clink	Slink	Proposed
yeast	20	20	21	21	20	21	22
glass	9	9	9	9	9	8	10
satimage	8	8	9	9	8	10	8
pendigits	20	20	22	22	20	20	22
optdigits	20	20	22	22	21	22	17
letter	58	58	65	65	63	66	69

Table 5.4: The number of nodes in the tree obtained on the UCI datasets by removing atmost 2 nodes at each node.

	I1	K-means	Maxcut	NCut	Clink	Slink	Proposed
yeast	0.0429	0.0429	0.0429	0.0429	0.0429	0.0429	0.0429
glass	0.1141	0.1141	0.1056	0.1056	0.1141	0.1141	0.1141
satimage	16.89	16.89	15.49	15.49	16.89	16.89	18.2421
pendigits	10.17	10.17	11.21	11.21	9.665	10.8	10.7963
optdigits	3.486	3.486	4.553	4.553	3.486	4.792	4.6267
letter	0.0295	0.0295	0.0269	0.0269	0.03	0.0366	0.0420

Table 5.5: The minimum margins obtained on the UCI datasets by removing atmost 2 nodes at each node.

	I1	K-means	Maxcut	NCut	Clink	Slink	Proposed
yeast	0.1433	0.1438	0.1408	0.1408	0.1468	0.1405	0.1442
glass	0.243	0.243	0.2503	0.2503	0.243	0.226	0.2473
satimage	23.43	23.43	24.22	24.22	23.43	24.2	24.5692
pendigits	17.2	17.2	17.51	17.51	16.53	16.5	17.0856
optdigits	6.009	5.979	6.042	6.042	6.011	5.866	6.0783
letter	0.0676	0.0676	0.0744	0.0744	0.0695	0.0722	0.0705

Table 5.6: The average margins obtained on the UCI datasets by removing atmost 2 nodes at each node.

	I1	K-means	Maxcut	NCut	Clink	Slink	Proposed
yeast	219.5	218.8	192.4	192.4	218.9	214.5	200.23
glass	36.56	36.56	37.33	37.33	36.56	40.5	35.10
satimage	577.6	577.6	563.1	563.1	577.6	522.3	523.50
pendigits	274.9	274.9	251.4	251.4	286.7	270.4	252.95
optdigits	132.6	133	125.4	125.4	132.5	131.1	125.71
letter	813.6	813.6	732.4	732.4	774.2	701.6	695.99

Table 5.7: The average number of support vectors obtained on the UCI datasets by removing atmost 2 nodes at each node.

5.7.2 Performance Evaluation

Graph based partitioning using different algorithms results in a variety of clusters depending on the objective functions and the distance measures chosen. Each cluster results in an initial margin, with a possibility to identify the bad classes and improve the margin.

5.8 Summary

In this chapter, we argue that no partitioning scheme is universally best for building hierarchical classifiers for all the data sets. We propose a locally optimal partitioning scheme to select the best partitioning algorithm at each node of the hierarchy. We propose a novel approach to build trees can be built with overlapping partitions. This results in improvement of performance with a small increase in size of the classifier are demonstrated on UCI datasets.

Chapter 6

Design of a Configurable Hierarchical Classifier

Many practical pattern recognition applications such as character recognition, biometrics etc, involve multiple classes of patterns to recognize. Binary classification is a well researched area whose theory can be extended for multiclass recognition. Few direct extensions of binary classifiers exist for multiclass classification. However, it is usually preferred to solve the multiclass classification using a combination of standard binary classifiers.

These classifier combination schemes can broadly be divided into two groups - two stage and hierarchical. Both of these approaches split the problem into multiple binary problems.

The most commonly used two-stage approach is the One-vs rest classifier combination. For an N class classification problem, one-vs-rest approaches build N classifiers, each with samples from one class as positive, and rest as negative. The decision of the most confident positive result of a classifier is assigned to the sample. The distributions of the positive and negative samples in this case are not compact and bounded, and hence the classifiers need to be highly complex, which results in poor generalization. These approaches are shown to be inferior to their more expressive counterpart – one-vs-one approaches in both accuracy, as well as training times. One-vs-one methods train $N(N-1)/2$ classifiers, one for each possible pair of classes. These are usually combined using voting approaches.

Hierarchical approaches usually connect the classifiers as a tree or a graph, and the result is obtained by following a path from the root node to the leaf node of the hierarchy. One approach to build a hierarchical classifier is to recursively partition the dataset, building a classifier to represent each partition. Another approach is to connect the pairwise classifiers in the form of a rooted binary Directed Acyclic Graph (DAG), called Decision DAG. Hierarchical classifiers have been popular for their intuitiveness in representation, and their speed. Hierarchical classification systems usually have much lesser classification time than the two-stage approaches as only a subset of the classifiers are evaluated.

The bound on the probability of error on unseen samples is called the generalization

Classifier	Combiner	Space	T_{best}	T_{avg}	T_{worst}
One vs Rest	Winner-takes-all	$O(N)$	$O(N)$	$O(N)$	$O(N)$
One vs One	Majority	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(N^2)$
	DDAG	$O(N^2)$	$O(N)$	$O(N)$	$O(N)$
Binary	Hierarchical	$O(N)$	$O(1)$	$O(N)$	$O(log(N))$

Table 6.1: The space and time complexities in best (T_{best}), average(T_{avg}), and worst (T_{worst}) cases in terms of number of classes for different architectures. First two columns specify the component classifier type and the combiner used.

error of a classifier. This is useful in analyzing the performance of a classifier. No such bound could be computed for the existing two stage approaches. Hierarchical classifiers are well in the scope of generalization theory and many analysis exist. Hierarchical classifiers are modular in nature, with a scope for independent designs for each module. This allows appropriate classifiers to be used in each module, resulting in a multiclass classifier with low generalization error.

The number of classifiers required and the time complexities for each of these multiclass designs is summarized in Table 6.1. The space complexity is measured in terms of the number of component classifiers built for combination. The time complexity is measured as the number of classifier evaluations to be made for the complete classification. Hierarchical classifiers with tree architecture has the least amount of space and time complexities, which is what makes them attractive for the practical applications.

Partitioning a dataset recursively results in a set of sub-problems which are dissimilar to each other in terms of complexity of classification. Using a uniform design for all the component classifiers does not suit all the sub-problems. Existing tree based architectures assume a homogeneous configuration for all the nodes in the tree. Instead, better trees can be induced by selecting the most appropriate classifier architecture for each partition.

In this thesis, we present a design for a hierarchical classifier system which maximizes its generalization, there by improving its performance. However, it is not sufficient if only performance is considered in the design, as size and time taken also form an important factor for various applications. The presented scheme incorporates constraints on size and time complexity into the design. The classifier design is applied to the problem of Optical Character Recognition (OCR), and its performance is studied.

6.1 Background

A multiclass problem can be divided into binary sub-problems in various ways. This is called binarization. The labels given to the samples in the binary problem (binarization) are called the meta-labels, and are different from their true labels in the multiclass problem.

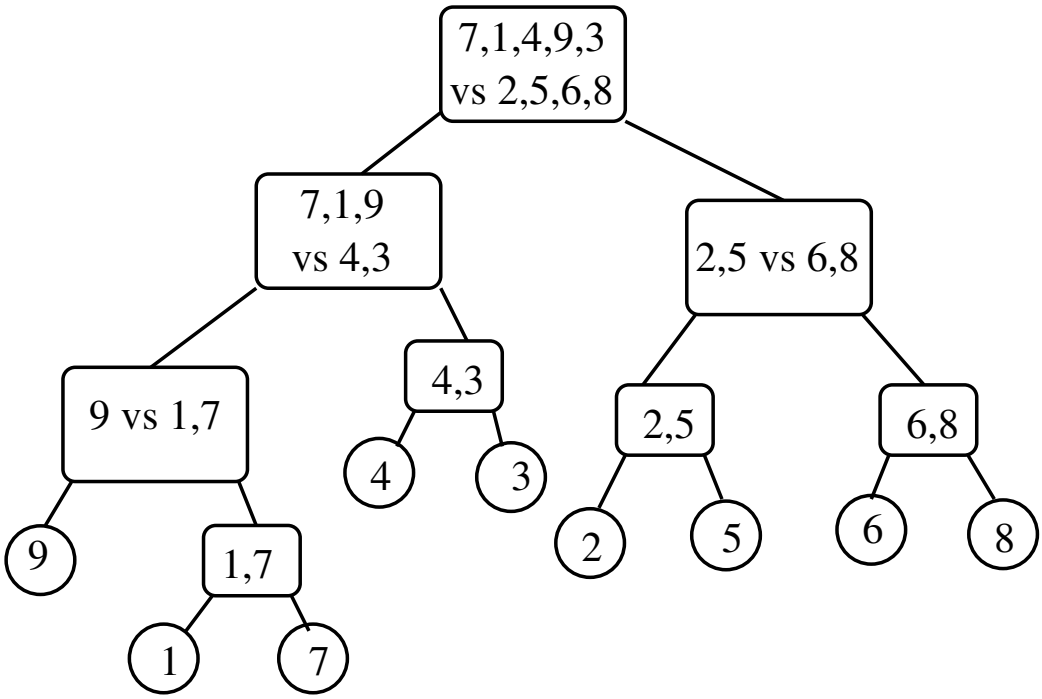

Figure 6.1: The BHCD classifier. The ovals are the binary classifiers for broad classification into groups of classes. The triangles are the DDAGs built to classify high-resemblance classes.

Popular binarizations are one-vs-one and one-vs-rest, as meta-labels are directly available in these cases. In one-vs-one, a DDAG has been a successful approach in combining the classifiers when compared to other voting approaches. Arbitrary binarizations have been used in approaches like ECOC and BHC. BHC has proven to be an efficient classifier in terms of size and speed.

BHC A BHC arranges $N-1$ binary classifiers in the form of a binary tree, such that there are N leaf nodes. An example tree for a digit classification task with 10 classes is shown in Figure 6.1. Usually in all the BHC algorithms, the dataset is recursively partitioned into two subsets at each step, and a classifier is learnt between them. The partitioning is done in a way that gives same meta label to all the samples from a particular class. Different clustering algorithms have been employed to partition the data into two clusters (eg. Local Linear Embedding [66], SOM [14], Kernel-SOM [14], K-Means [82], Spherical Subsets [82]). A classifier is trained between the samples of these meta-classes. A BHC has $N-1$ nodes and runs on the average $O(log(N))$ classifiers on a sample. Different approaches have been tried out using overlapping and non-overlapping subsets of classes.

DDAG A rooted binary DDAG was used in [67] to combine the pairwise classifiers. In a rooted binary DDAG, each node has either 2 or 0 children, and there is a unique node without a parent, which is called the root. A DDAG is evaluated on a given sample in the following way. The sample is presented to the root node, and the binary function at the root node is evaluated. Depending on the decision at the root node being positive or negative, left or right child node is chosen correspondingly. The decision function at this node is then evaluated to select its children. This continues until the sample reaches a leaf node where the sample is assigned a class label. The path which the sample has taken in reaching the leaf node is called the *evaluation path*. At each classification, one class is rejected and the sample moves to the sub-DAG containing $N-1$ classes. An example DDAG for six digits is shown in Figure 3.1.

Comments Unlike popular multiclass approaches, which test a given class against all the other classes, BHC tests a class against subsets of classes, resulting in exponentially faster classifications. In two-stage classifier combination approaches, similar set of features are used usually for all the classifications. A BHC provides modularity, where each binary problem in the tree can have its own set of features depending on the complexity of the classification task. This flexibility provides improvement in performance, as appropriate features are used for each classifiers instead of a global set of features [54]. In a BHC, an error committed at the root may not be correctable at any further stage of classification. This way, uncorrectable errors accumulate at each level, and reduce the performance of the BHC. However, in a DDAG, the binary classifications are not as critical as in the case of a BHC. This can be explained as follows. A classifier trained on certain classes is called a *relevant-classifier* to those classes, and otherwise *irrelevant classifier*. Any hierarchical classifier makes an error in the final decision, only if a relevant classifier makes an error. In a BHC, although there are lesser number of classifiers on the evaluation path, all of them are relevant, which is not true in the case of a DDAG.

BHC is very effective in terms of size and classification speed, but the arbitrary binary classification problems at the nodes might be complex affecting the performance. DDAG on the other hand, is a multiple classifier approach, which exploits the redundancy in the pairwise classifiers to obtain a robust classification. The pairwise classifications are relatively simple when compared to the binarizations in the BHC, which may be complex depending on the partitioning algorithm used. DDAG uses $O(N^2)$ classifiers, and the size can be prohibitively large when addressing problems with large number of classes.

A BHC is efficient in terms of speed and size, and a DDAG in terms of accuracy. This has been empirically verified in [82]. However, no classifier is superior to other in all the cases. Hierarchical classifiers are to be designed by selecting the most appropriate classifier and architecture at each level. The proposed algorithm combines the advantages of BHC and DDAG resulting in a hierarchical classifier with hybrid architecture.

The rationale in the proposed algorithm is as follows. A BHC on average, rejects half of

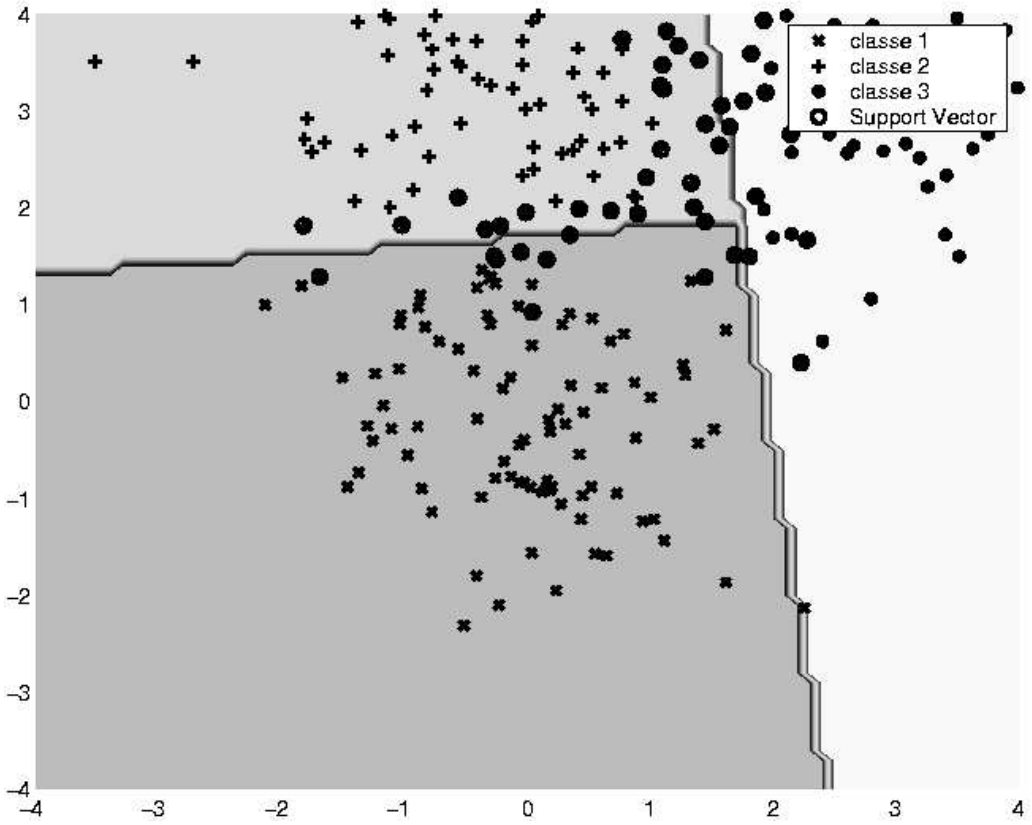

Figure 6.2: The decision boundaries obtained by a BHC classifier for a 3 class case.

the available number of classes at each stage of classification. However, when the classifications are complex, this might affect the performance of the classifier. In these cases, using a DDAG to reject a single class at each stage, with a possibility of error correction up to a certain extent is an alternative.

Example: The way BHC and DDAG operate on a 3 class dataset are shown in Figure 6.2 and Figure 6.3, respectively. The samples from the three classes are follow normal distributions with three different means and variances. In this particular example, the DDAG performed better than a BHC. In the figures, bottom region shows class 1, top region shows class 2 and right region shows class 3. Considering two classes at a time, the decision boundaries between 1,3 and 2,3 are optimal in the case of a DDAG where as they are not optimal in the case of a BHC. This implies that between these two discriminations, a BHC makes more error. The decision boundary between 2,3 is same in both the cases. Therefore, the overall error obtained by a BHC is higher than a DDAG in this case, which was verified experimentally.

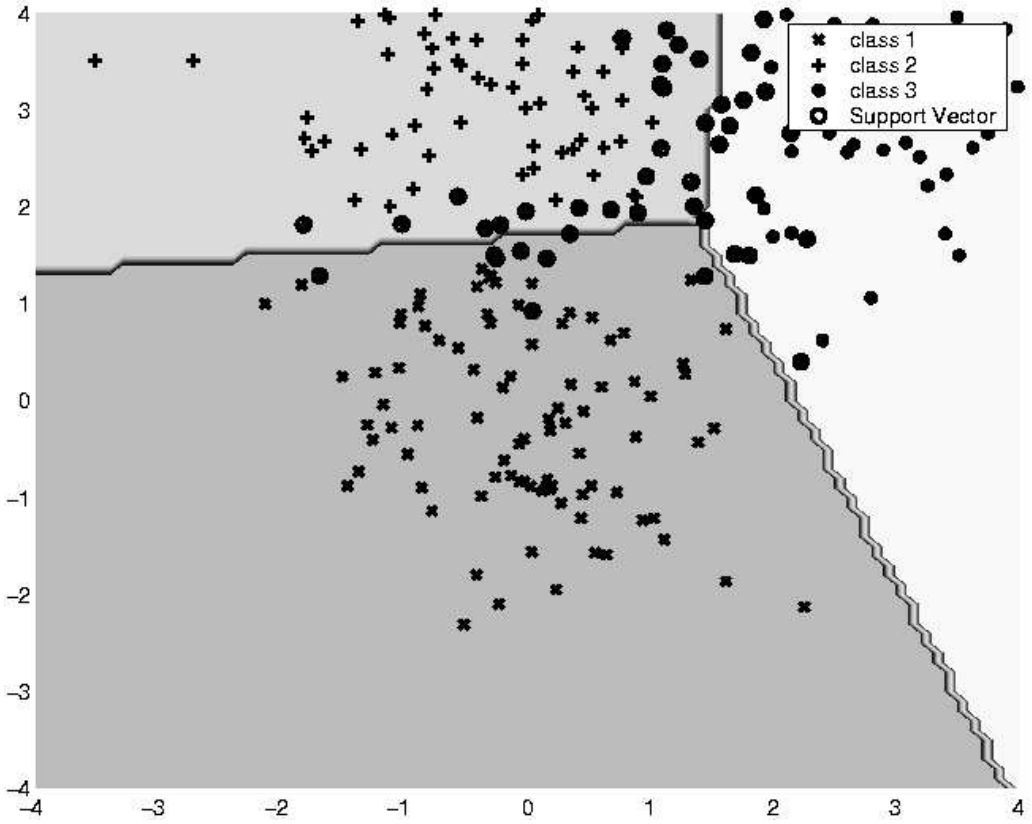

Figure 6.3: The decision boundaries obtained by a DDAG classifier for a 3 class case.

Analysis The ability of a classifier to generalize what it has learnt from the training data over unseen samples is called generalization. For a hierarchical classifier, the generalization depends on the number of training samples m, the depth of the evaluation path T, and the margins obtained for the binary classifiers on the evaluation path. For the generalization ability of a classifier $G(\epsilon)$, with a probability $1-\delta$ it is true that

$$G(\epsilon) \leq \frac{130R^2}{m} D' log(4em) log(4m) + log \frac{2(2m)^T}{delta}$$

where $D' = \sum_{i=1}^{T} \frac{1}{\gamma^2}$.

Let $L(m) = \frac{log(4em)log(4m)}{log(2m)}$. The above equation can be rewritten as

$$\begin{aligned} G(\epsilon) &\leq \frac{130R^2}{m} D' L(m) log(2m) + log\frac{2}{\delta} + T log(2m) \\ &= \frac{130R^2 log(2m)}{m} (D' L(m) + log\frac{2}{\delta} + T) \end{aligned} \tag{6.1}$$

For a dataset, since m, R are same for both the algorithms, fixing a δ, Equation (1) results in

$$G(\epsilon) \propto (D' L(m) + T)$$

. L is a monotonically increasing function of m. The following can be observed from the above equation

- The performance of the hierarchical classifier depends on the amount of training data m, the length of the evaluation path T and the margins obtained γ.
- The length of the evaluation path is high in a DDAG, however is compensated by large margins obtained for the pairwise classifiers.
- Since D' is defined in terms of margins, as the number of training samples m increase, the effect of margins is more on the generalization, compared to the length of evaluation path T. When the training data is more, margins are the dominating factor. Preferring a method which gives higher margin results in lesser classification error.
- When the training data is sparse and the classes are very large in number, using a BHC would result in better performance.
- The margins in a DDAG are high when compared to a BHC. As long as the difference between T_{BHC} and T_{DDAG} is not large enough to affect the margin term, using a DDAG is preferable.

When T is large as in the case of a DDAG, the D' is small, and when T is small as in the case of a BHC the D' term is large. The better performance of a DDAG or BHC depends on the interactions of all these parameters, and hence none of them is absolutely superior

over the other approach in all the cases. This shows that an algorithm which combines these two architectures must result in appropriate values of the T and margins resulting in classifiers with better performance. We present a hybrid algorithm to evolve a hierarchical classifier by choosing the appropriate architectures at each step.

A brief overview of the hierarchical classification methods is presented in Section 2. The problem is formulated in section 3, and an algorithm called the Binary Hierarchical Combination of DDAGs is presented in Section 4. Performance of these classifiers is analysed in section 5. Results on two large class problems are presented in section 6.

6.2 Hierarchical Classifiers for Pattern Classification

This section presents an overview of the existing hierarchical techniques used for pattern recognition.

6.2.1 Feature Based Hierarchies

These are the class of hierarchical classifiers in which each decision making node is a function of a subset of features used to represent the samples. These functions are either specified manually or can be learnt from the data as in the case of decision trees.

Manual Hierarchies The hierarchy in these systems is determined manually based on perceptual similarity of the samples, or using simple cues. In effect, it is a hierarchal representation of man made rules, which functions as an expert system. Such hierarchical classifiers are being used for large-class OCRs in [58] and in [5, 12] for Oriya, Bangla and Devangari character recognition systems. Although they are quick for being hierarchical and having a hand-picked, minimal and apparently best set of rules, they are very limited in their scope for the following reasons. The hierarchy is not derived from the structure of the problem by using the training data available. Instead they are an encoding of a set of rules given by a single expert, with his knowledge of the problem. Provided more training data, these systems cannot improve themselves. A high correlation had to be maintained between the perceptual similarity of the objects and their feature representation in order to obtain hierarchies which give good performance. In the case of systems like OCRs, these rules make system specific to a set of fonts or styles.

Decision Trees Instead of building the rules manually based on features, learning them is always preferable. This makes the algorithms independent of feature representations, and uses the training data to come up with a hierarchy. Decision trees [68] are classical approach to attempt a hierarchical representation of the classification knowledge. They were popular for organizing data and efficient searching as the number of comparisons required is of logarithmic order of the number of classifiers, where as it turns out to be linear or quadratic

in the non-hierarchical approaches. Different architectures were proposed for decision trees of which univariate, multivariate, perceptron and omnivariate decision trees were popular. The complexity of the classifier at a node distinguishes these architectures from each other. First three approaches have a fixed design for each node, where as omnivariate decision trees choose the best design for each node. Omnivariate trees thus resulted in better classification accuracies than their homogeneous counterparts. This is because, using the classifier of right complexity at each node is what results in the best generalization, and omnivariate trees achieve that. However, these approaches attempt to build a tree representing the rules at the feature level. However, they are highly affected by noise in the data, resulting in very large trees.

6.2.2 Class Based Hierarchies

These are class of hierarchical classifiers where the data set is partitioned into multiple subsets at each decision making node depending on the classes of samples. These are usually of the linear order of the number of classes, which is very small when compared to feature based hierarchies which grow at an exponential order of samples in the case of difficult classification. Few such algorithms are described below.

Decision Directed Acyclic Graphs This is an efficient way to combine the pairwise classifiers, to overcome the quadratic evaluation time of the pairwise majority classifier [67].This connects the pairwise classifiers as a rooted binary directed acyclic graph. The sample is initially presented to the root node of the graph, and depending on the decision taken, either left or right sub-DAG is selected for next evaluation, until a leaf node is reached. The sample is assigned the label of the leaf node it reaches. The classification time is significantly brought down to $O(N)$ from $O(N^2)$ of pairwise majority. However, the size of the classifier still remains huge. This architecture explores the redundancy existing in pairwise classifiers to result in high performance, which is lacking in other hierarchical approaches. The architecture is fixed for a given set of classes irrespective of the training data and its feature representations.

Automatic Learning of Hierarchies Automatic learning of class hierarchies, resulting in a binary tree architecture is a powerful approach for multiclass classification. The size of these classifiers is of $O(N)$ and the testing time is $O(logN)$. Binary Hierarchical Classifier (BHC) [54, 13] is one such method, which learns the best partitioning of the classes into two subsets at each node of the tree using a deterministic annealing algorithm. It builds $N-1$ classification nodes for an N class problem. DB2-SVM [82] attempts to build a hierarchical SVM using sample level clustering at each node. At each node, a clustering algorithm divides the classes in the data into two subsets, and an SVM is built. Although these algorithms are extremely efficient in time and space, they sometimes compromise on

accuracy. Each node in this architecture is evolved depending on the training data, the architecture is still fixed for the overall tree. The tree is homogeneous in nature, which requires the same classifier to be used at all the nodes irrespective of the generalization issues. A classifier design should hence consider the following issues

- The generalization of the classifier at architecture level has to be accounted for
- The time taken to classify should be as small as possible, and be configurable in the case of availability of more time.
- The size of the classifier built should be as small as possible, and rather be configurable so that better classifiers be built on availability of more space

In this thesis, we address these problems and propose a method to design a configurable hybrid architecture.

6.3 Problem Formulation

The problem of supervised classification is to assign the class label ω_i to a given sample $\mathbf{x}_m \in \mathcal{X}, m = 1 \cdots M$, where $\omega_i \in \Omega, i = 1 \cdots N$ are the classes labels for a set of N classes. A hierarchical classifier for supervised classification is composed of multiple nodes connected in the form of a tree. The number of nodes K and the way they are combined depends on the architecture used. Let $\Theta = \{\theta_k\}, k = 1 \cdots K$ be the set of classifier parameters, with θ_k corresponding to the kth node. Let $f : \mathbf{x}, \mathbf{\Theta} \rightarrow \mathbf{\Omega}$ denote the combiner, which evaluates the node on the sample and predicts the class label. Let t_m be the true label of mth sample. Since hierarchical classifier is a tree based classifier, it is induced top-down, as the problem of building an optimal decision tree is NP-hard. Let f_i denote the node that is being built now, and f_{i-1} denote its immediate parent. A usual approach followed to learn the combiner $f(\mathbf{x}, \mathbf{\Theta})$ is to minimize the empirical error J_{emp} at each step, defined as

$$J_{emp}(\mathcal{X}, f_i) = \frac{1}{M} \sum_{m=1}^{M} \delta(t_m, f(\mathbf{x}_m, \Theta))$$

where $\delta(t_m, f(\mathbf{x}_m, \Theta)) = 1$ if $t_m = f_i(\mathbf{x}_m, \Theta)$, and 0 otherwise. Algorithms minimizing J_{emp} attempt to find the architecture f and node parameters θ_k that minimize the error on the training data. This results in the problems of overfitting and poor generalization [81].

Large margin classifiers [81] are successful in building classifiers with good generalization performance. Using them one can achieve a node level generalization by including a margin maximizing term at each node along with its empirical training error. Let $J(\mathcal{X}, f_i)$ represent the objective function for obtaining node level generalization,

$$J(\mathcal{X}, f_i) = J_{emp}(\mathcal{X}, f_i) + \rho_k \sum_{k=1}^{K} C(\theta_k)$$

where, $C(\theta_k)$ is a cost term, which increases as the complexity of the classifier increases. related stuff It acts as a trade off between the classifier complexity and the training error. Hierarchical SVMs [13, 82] tend to minimize this objective function. A weighting factor ρ_k is used to specify a balance between cost of misclassification and the complexity of the classifier. It lacks in incorporating any parameters related to the architecture although the problem demands it.

In the case of improving the classifier at node level, the objective is always to obtain the best performance in terms of accuracy. Where as, in the case of architecture, the objective could change with the requirements. Different classifier systems can have different constraints in terms of speed, robustness, accuracy, parameter size, etc. Therefore a cost term $C_{arch}(f, m)$ has to be included in the formulation, whose behavior depends on the requirements of the system. Including it in the formulation would result in the objective function J_{gen},

$$J_{gen}(\mathcal{X}, f_i) = J(\mathcal{X}, f_{i-1}) + \lambda C_{arch}(f_i, m) \tag{6.2}$$

The parameter λ acts as the weighting term between the architecture complexity and the generalization of the classifier. For example, C_{arch} can be chosen in such a way that it increases with the increase in number of nodes in the classifier. In this case, if λ is high, smaller number of nodes are preferred even at the cost of a lower accuracy. If λ is low, the resulting architecture is closer to the ones obtained using approaches minimizing $J(\mathcal{X}, f)$. Therefore, λ can be used to control the size of the classifier in this case. Similarly, other objectives like shorter architectures, fast classification rate can be obtained by appropriately defining the C_{arch}, which can be further controlled by choosing an appropriate λ.

6.4 Hybrid Hierarchical Classifier

A binary hierarchical classifier connects several two class classifiers in a hierarchy such that a sample is classified in a series of steps. The classification starts with broad and simple decisions and narrows down in stages to a precise decision at the leaf. Each node divides the set of available classes into two groups. However, all the classification tasks at the component classifiers are not equally complex. Many a time, binarization can not be done effectively for all possible subsets of classes, where a more complex multiclass classifier is required.

The hybrid classifier is a hierarchical arrangement of dissimilar classifiers, each chosen according to the complexity of the classification task. The current algorithm builds a hybrid classifier using two kinds of component classifiers, a linear binary classifier for simple classifications and a DDAG for complex multiclass groups. A DDAG is a high-performance classifier, but the number of component classifiers increase rapidly with increase in number of classes. The BHCD algorithm hence uses DDAG only on smaller subsets of classes which are difficult to classify within each other. A BHCD for a set of Devanagari characters is

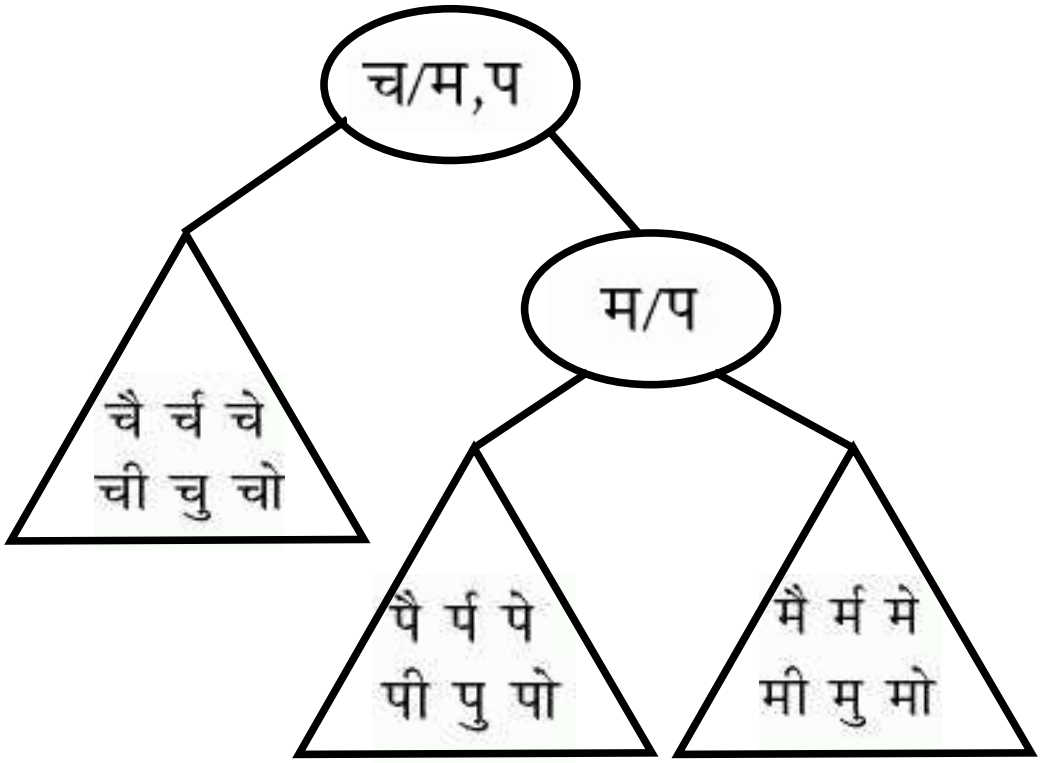

Figure 6.4: The BHCD classifier. The ovals are the binary classifiers for broad classification into groups of classes. The triangles are the DDAGS built to classify high-resemblance classes.

shown in Figure 1. The ovals are the binary classifiers between groups of classes, which are easy to classify. DDAGs are used to perform the within class classification of these groups, and are shown as triangles in the Figure 1.

6.4.1 BHCD Algorithm

A recursive algorithm is proposed to learn the hierarchical classifier. At each step of creating a node in the hierarchy, a selection is made between a binary classifier and a DDAG. For the binary classifier, a clustering algorithm is used to split the dataset into two subsets. The difficulty of classification of this binary classification is estimated. Of these two, the classifier with greater expected accuracy is chosen. To select the appropriate classifier, a "goodness" measure resembling its error estimate is needed. Section 6.4.2 describes the measure used in the algorithm.

A DDAG is a multiple classifier method, which performs well due to the redundancy provided by the pairwise classifiers. Error estimates are always low for a DDAG because of the simplicity of pairwise classifications when compared to arbitrary binary classifiers. If a

DDAG is always preferred, the resulting classifier suffers from large size and classification time. To discourage the selection of DDAG always, a parameter λ is used to weigh the advantage obtained while selecting the component classifiers at each node. The user provides the parameter λ to specify the relative importance of accuracy to the size and classification time of the hierarchical classifier.

Let f denote the the hybrid classifier with K nodes, and $\theta_i, i = 1 \cdots K$ be the parameters of the classifier at the ith node. At each node i, $\Omega^{(i)}$ denotes the set of labels of the classes that reach that node, Let $\Omega_l^{(i)}, \Omega_r^{(i)}$ represent the set of classes that correspond to the left subtree and right subtree respectively, $\mathcal{X}^{(i)}$ represent the samples reaching the node i and the operator $\oplus$ be used to denote the addition of a set of parameters at the current node to the existing tree. The functions *trainBinary* and *trainDDAG* are assumed to be the training algorithms for binary classification and a DDAG respectively. Using this notation, Algorithm 8 describes the procedure formally.

Algorithm 8 BHCD($\mathcal{X}, \lambda$)

$f = \phi$
$(\Omega_l^{(i)}, \Omega_r^{(i)})$ = Cluster($\mathcal{X}_i^{(i)}, 2$); // Make two clusters
$\mathrm{L}_{binary} = ComputeClassifiabilityEstimate(\mathcal{X}_i^{(i)}, l, r)$
$\mathrm{L}_{ddag} = ComputeClassifiabilityEstimate(\mathcal{X}_i^{(i)}, \Omega_i)$
if $(\mathrm{L}_{ddag} < \lambda \mathrm{L}_{binary})$ **then**
  $f = f \oplus \theta_i$
  θ_i = trainBinary($\mathcal{X}_l^{(i)}, \mathcal{X}_r^{(i)}$)
  if $(n(\Omega^{(i)}) > 2)$ **then**
    BHCD($\mathcal{X}_l, \lambda$)
    BHCD($\mathcal{X}_r, \lambda$)
  end if
else
  θ_i = trainDDAG($\mathcal{X}^{(i)}$)
  $f = f \oplus \theta_i$
end if

6.4.2 Classifier Evaluation

A key step in the algorithm is to compute the "goodness" of different classifiers at each step of tree building, and select the suitable classifier. We use an adapted version of the classifiability measure proposed in [18] for computing the possible error estimate on the datasets. The measure is closely related to the Bayesian error, and is easily computable. To start with, define an appropriate neighborhood size r. For each pattern $x^{(i)}$ which belongs to class $\omega_l \in \Omega$, obtain a co-occurrence matrix $W(x^{(i)})$ of size $N \times N$, whose element $w(x_{jk}^{(i)})$

is defined as $w(x_{jk}^{(i)}) \equiv \sum_{m=1}^{N_{\omega_k}} f(x^{(i)}, x^{(m)})$, where $x^{(m)}$ is a sample of class ω_k. $f(.)$ is 1 if $||x^{(l)} - x^{(m)}|| \leq r$ and $j = l$, 0 otherwise.

Compute the overall coocurrence matrix [18] as the sum of coocurrence matrices of all the samples, i.e $A = \sum_{m=1}^{M} W(x^{(m)})$. Normalize the elements a_{ij} of A such that sum of all the elements is 1. Then the classifiability L can be computed as,

$$L = \sum_{i=1}^{c} a_{ii} - \sum_{j \neq k}^{c} a_{jk}.$$

However, this measure computes the classifiability of all the classes together. The classifiability of datasets is different when pairwise subsets are considered. Here, we adapt this measure to provide an error estimate of the pairwise classifier, by taking average of pairwise classifiabilties. Let L_{lm} be defined as the classifiability between classes ω_l, ω_m. The pairwise classifiability L_{pw} is defined as

$$L_{pw} = \sum_{l,m \in \Omega^{(i)}, l \neq m} L_{lm}.$$

If $L_{ddag} - \lambda L_{binary}$ is negative, the binary classifier is used at the node. Otherwise, a DDAG is built between all the classes at that node.

6.5 Analysis

Generalization ability and computational complexity are important parameters to measure the effectiveness of a classifier. An analysis of the BHCD algorithm from these perspectives is presented in this section.

6.5.1 Generalization

For a classifier to perform well on unseen samples, the learning algorithm has to reduce the generalization error of the classifier. Many of the Asian and African scripts are characterized by groups of classes spread around the feature space. One group of classes is distinctly separate from the others, but the classes within a group resemble each other and are difficult to classify. If a simple classifier is used for all the classes together, the classifier underperforms for the within group classification. If a very complex classifier is used, it performs well within the groups, but over-fits and takes large space and classification time. Since, BHCD algorithm evolves the hierarchy by selecting the classifier of required complexity at each level of the hierarchy, it generalizes well.

The path taken by the sample from the root node to the leaf node of a hierarchy while being classified is called the *evaluation path*. In [6], it was shown that the generalization error of a tree classifier depends on the length of the evaluation path T and the margins of nodes on the path. According to [82], the best case performance of a tree based classifier

	$\lambda = High$ (BHC)			$\lambda = Medium$ (Hybrid)			$\lambda = Low$ (DDAG)		
	Accuracy	Storage	T_{eval}	Accuracy	Storage	T_{eval}	Accuracy	Storage	T_{eval}
Tamil	91.50	119	8	93.6	1029	26	93.4	7140	119
Malayalam	98.2	115	8	98.62	990	24	98.60	6555	115
Amharic	84.72	224	9	86.82	2250	31	87.20	92235	224
Letter	71.20	25	6	73.23	105	8	75.16	325	25

Table 6.2: Accuracies of classifiers built with varying λ. The size of the classifier (storage) is shown as the total number of nodes in the classifier. The evaluation time of the classifier (T_{eval}), measured as average depth of the tree.

where $T = 1$, is better than a DDAG, where $T = N - 1$ and both classifiers are equivalent in the worst case. When there are large number of classes, a BHC has a lower generalization error bound than a DDAG, owing to the difference in the lengths of evaluation paths [82]. At the deeper nodes of a hierarchical classifier, the difference between the lengths of evaluation paths between a DDAG and a BHC is less as there are fewer number of classes. However, at this stage, the pairwise margins obtained in a DDAG are higher than the margins obtained by an arbitrary binary classifier in BHC. This shows that the BHCD algorithm selects classifiers to reduce the overall generalization error of the hierarchical classifier.

6.5.2 Complexity

Table 6.1 presents the orders of space and testing time complexity in classifiers for various popular classifier combination architectures. The testing time complexity, expressed in terms of number of classifiers evaluated for each decision to be made is denoted using C. As an example, consider an OCR for an example Indian language say, Telugu. It has approximately 430 classes to be recognized. Using a one-vs-one training with a DDAG or a majority vote combiner, and stores around 92,235 classifier parameters, which is prohibitive. Using a majority vote based approach here would result in evaluating all the classifiers at least once per sample. Whereas, in a hierarchical classifier, there are only 429 classifiers built, of which on average 7 classifiers get evaluated. This gain in space and time are immense, even at the cost of a little reduced accuracy, depending on the requirements of the application.

6.6 Results and Discussion

6.6.1 Optical Character Recognition

Optical Character Recognition (OCR) applications range from massive tasks like digitization of large document databases, to tiny applications in a PDA or a cell phone for

text access while being mobile. Each of these applications have different requirements and constraints on parameters like the size, speed and accuracy. English has smaller set of characters which makes it easy to build OCR applications of different sizes and performances. However, this was not possible with Asian and African scripts owing to the large size of their alphabet.

In many of the Asian and African scripts, the characters are formed by conjunctions of basic shapes. Owing to the large number of possible combinations, these scripts have a very large number of characters. Moreover, the characters which share one or more basic shapes are highly similar to each other, making the classification task further difficult. Building a classifier to recognize all the characters independently is not possible as the resulting classifier would be prohibitively large. Many of the characters can be segmented into their basic shapes. However, this only reduces the number of characters to a lesser number of classes to recognize. Table 6.6.1 shows the details of few Asian and African scripts in the form of the number of basic alphabet, the number of characters that arise from their combinations, and the approximate number of classes that are used in a typical character recognizer.

Hierarchical classifiers [2, 58] are found to be suitable for large class classification, as they have low space and time complexity. They are highly modular in nature, allowing use of a variety of classifiers at each node. However most existing designs do not explore this modularity, and use similar classifiers at all the nodes [54, 58]. This results in classifiers which under-perform, as the classification problems at each node are quite different from each other. An effective way to build a classifier is to evolve the hierarchy by choosing appropriate classifiers at each node in the hierarchy. The appropriateness of a component classifier can be measured in terms of speed, size and accuracy, depending on the requirements of the system.

Language	Alphabet	Characters	Classes
Devanagari (India)	57	64000	165
Telugu (India)	56	64000	432
Hangul (Korea)	67	11772	2350
Hiragana (Japan)	46	5000	1945
Amharic (Ethiopia)	34	310	225

Table 6.3: Few Asian and African scripts, the number of basic alphabet, possible characters and classes in a typical OCR system.

Experiments are conducted on character recognition problems. Four datasets were used in the experiments, two Indian language scripts: Tamil and Malayalam, an Ethiopian script – Amharic and the Letter dataset from the UCI machine learning repository. Tamil dataset has 120 classes, Malayalam has 116 classes, and Amharic dataset has 225 classes. Each of

them have 100 samples per class. Letter dataset has 26 classes, and 20000 samples. In each experiment we used 60% of the data for training and rest for testing. We used a linear SVM as the classifier at all nodes of the hybrid classifier built including the pairwise classifiers in the DDAG. Results show that BHCD performs better than BHC always, and sometimes better than the DDAG, taking significantly lesser space than a DDAG, on par with a BHC.

UCI Letter Dataset: UCI Letter dataset [40] has 26 classes. In this case, a DDAG requires 325 classifiers to be built and a BHC requires 25 classifiers. The number of classifier evaluations made when using a DDAG is 25, and using a BHC is around 5. Using linear SVMs at each node, the accuracy obtained with a DDAG is 75.16% and with a BHC is 71.20%. The BHCD algorithm is found to improve the accuracy of the classifier over BHC, and reduces the storage space by 3.1 times and classification time by 4.1 times compared to a DDAG.

Malayalam Character Recognition: Malayalam, a South Indian language is considered for this experiment. The number of classifiers required for a DDAG is 6670, and makes 115 classifier evaluations to make a decision. The accuracy obtained using a DDAG is 98.60%. The number of nodes in a BHC is 115, and around 7 classifier evaluations are necessary to classify a given sample. The accuracy obtained by a BHC is 98.20%, which is better than both the approaches and reduces the storage space by 6.5 times, and classification time by 4.8 times with respect to a DDAG.

Tuning the Classifier: Experiments on large class datasets are conducted to show the effect of the control parameter λ on the design of the classifier. Hybrid classifiers were generated with varying values of λ and the accuracies and size values of the classifiers are computed. We grouped the λ values into three groups – low, medium and high. For each dataset, the accuracy, storage and the evaluation time are presented in Table 6.2. When λ is high, the resultant classifier is a binary hierarchical classifier. When λ is low, there is no constraint on the size, and hence a classifier DDAG. When λ is medium, a classifier of size larger than the tree, but much smaller than a DDAG is obtained, with performance higher than the tree in all the cases, and also higher than DDAG in some cases. A large increase in the classifier size as well as the depth of the tree in the classifier was observed as the value of λ decreases. In most of the cases, the drop in the accuracy observed was a minimal compromise for the large reduction obtained in size and testing time.

6.7 Summary

An algorithm for building hybrid classifiers combining the advantages of two algorithms BHC and DDAG is presented. It is shown that using a hierarchical combination of complex

classifiers, suits the problem of large class character recognition. This results in classifiers with better generalization and with less space and time complexities.

Chapter 7

Conclusions

Pattern classification problems with large number of classes are difficult to solve with direct methods. A popular promising alternative is to build large number of smaller, accurate and efficient classifiers, and thereafter to integrate their decisions. Popular integrating techniques like majority voting are not appealing because of their computational (evaluation) complexity. Decision Directed Acyclic Graphs (DDAGs) and Binary Hierarchical Classifiers (BHCs) are popular option. However, their design for improving the accuracy beyond the simple direct integration is not addressed in literature. This Thesis proposes many design algorithms for these two classifiers.

Feature extraction and classifier design form the major components of any classifier system. In this thesis, we have addressed both issues, analytically and empirically. An improvement at the node-level is made to the classification system in the design of a DDAG and features selection. Optimal discriminant features for pair-wise class combinations are extracted for use with the DDAG classifier. This suits the pairwise SVM classifier we used at each node of the DDAG. However, using pairwise features introduces a lot of cost in both space and time for classification. To overcome this, a method involving linear discriminant analysis followed by PCA for dimensionality reduction is proposed. This setup solves the problem of space complexity encountered by using PCA followed by LDA. For improving the classifier, Boosting the DDAG at node level is done. With this, we registered high accuracy with less space and time complexity. Furrther improvement to the DDAG classifier is also proposed. The basic premise is that the arrangement of the nodes in the DDAG affects the performance of the complete multiclass classifier. A novel formulation for the design of multiclass classifier is achieved based on the rearrangement of order of nodes. Algorithms for designing better multiclass classifiers using the DDAG architecture are formulated. The algorithms proposed are approximate, as the problem of building optimal DDAG is a permutation selection problem, and is difficult to solve in polynomial time.

Although DDAG is a robust classifier with high accuracy, there are a few limitations to the DDAG in terms of size and classification time. If the dataset has a large number of

classes, the number of classifiers in a DDAG gets prohibitively large. A possible alternate is to use the Binary Hierarchical Classifier (BHC), which has very low space and classification time complexity when compared to a DDAG. Algorithms for designing improved BHCs are suggested. Also it is concluded that no partitioning scheme is universally best for building hierarchical classifiers for all the data sets. An algorithm which takes advantage of many existing partioning schemes, and selects a locally best partition at each node based on the margin obtained is proposed. A novel scheme of building trees with overlapping partitions is introduced. This resulted in improvement of performance of the hierarchical classifier with a small increase in size of the classifier. Extending this, a hybrid design of a classification system, which exploits the advantages of different hierarchical architectures is presented. A classifier called Binary Hierarchical Combination of Decision Directed Acyclic Graph (BHCD) is proposed. This uses a greedy algorithm to select a binary partition or a direct DDAG at each node of hierarchy building. This algorithm is shown to have the time complexity close to the BHC, while being as accurate as the DDAG.

This thesis describes algorithms for designing hierarchical classifiers. Class-based hierarchies are learned from the training data or the probabilistic structure of the performance of the individual nodes. Analysis of complexity of the training process, feature selection methods and design algorithms are reported. Numerical results on synthetic data sets and standard data sets are presented. Special emphasis has been made on the applicability of these methods for the character recognition problems.

7.1 Comments and Future Work

- It is argued that the design of hierarchical classifiers is not attempted from an algorithmic perspective in literature. With the help of innovative yet simple design schemes, the performance of the high-performance classifiers can still be improved.

- For the hierarchical classifiers, performance of the nodes can be independently improved using appropriate feature selection techniques. We also make sure that this does not introduce any additional computational complexity. Algorithms for design of fixed architectures (eg. DDAG) can work on class ordering to improve the performance.

- Binary Hierarchical Classification needs modules for partitioning the classes into two mutually exclusive subsets. We argue that, by allowing overlapping partitions, we can build better nodes. Also the nodes can be of heterogeneous in nature to improve the performance.

- It is shown that the design process can help in improving the accuracy of the high-performing classifier systems. Feature-based hierarchies were studied in detail in the

context of decision trees. An integrated design of the feature and class based hierarchies will be of immense help in many pattern recognition problems. This is an unexplored direction of promising research.

- Many problems in computer vision and many other disciplines of information technology need efficient and accurate classification of patterns. There algorithms can improve the effectiveness of the overall performance of the computer vision problems.

Bibliography

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

[2] H. Baird and C. Mallows. Bounded-error preclassification trees. In D. Dori and A. Bruckstein, editors, *Shape, Structure and Pattern Recognition*, pages 100–110. World Scientific Publishing Co., 1995.

[3] S. Baker and S. Nayar. Pattern rejection. In *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1996.

[4] Veena Bansal and R. M. K. Sinha. A devanagari OCR and a brief review of OCR research for Indian scripts. In *Proc of STRANS01*, 2001.

[5] B.B.Chaudhuri, U.Pal, and M.Mitra. Automatic recognition of printed Oriya script. *Sadhana*, 27(1):23–34, February 2002.

[6] Kristin P. Bennett, Nello Cristianini, John Shawe-Taylor, and Donghui Wu. Enlarging the margins in perceptron decision trees. *Machine Learning*, 41(3):295–313, 2000.

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

[8] L. Brieman, J. Friedman, R. Oshlen, and C. Stone. *Classification and Regression Trees.* Wadsworth International Group, 1994.

[9] C. V. Jawahar, M. N. S. S. K. Pavan Kumar, and S. S. Ravi Kiran. A Bilingual OCR for Hindi-Telugu Documents and its Applications. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 408–412, 2003.

[10] Lijuan Cai and Thomas Hofmann. Hierarchical document categorization with support vector machines. In *Proceedings of the thirteenth ACM conference on Information and Knowledge Management*, pages 78 – 87, 2004.

[11] Michelangelo Ceci and Donato Malerba. Hierarchical classification of html documents with webclassii. In Fabrizio Sebastiani, editor, *Proc. of the 25th European Conf. on*

Information Retrieval (ECIR'03), volume 2633 of *Lecture Notes in Computer Science*, pages 57–72. Springer, 2003.

[12] B. B. Chaudhuri and U Pal. An OCR system to read two Indian language scripts: Bangla and Devanagari (Hindi). *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, pages 1011–1015, 1997.

[13] Y. Chen, Melba M. Crawford, and J. Ghosh. Integrating support vector machines in a hierarchical output space decomposition framework. In *IEEE International Geoscience and Remote Sensing Symposium, Alaska AK*, volume 2, pages 949 – 952, September 2004.

[14] Sungmoon Cheong, Sang Hoon Oh, and Soo-Young Lee. Support vector machines with binary tree architecture for multi-class classification. *Neural Information Processing - Letters and Reviews*, 2(3):47–51, March 2004.

[15] Wesley T. Chuang, Asok Tiyyagura, Jihoon Yang, and Giovanni Giuffrida. A fast algorithm for hierarchical text classification. In Yahiko Kambayashi, Mukesh Mohania, and A. Min Tjoa, editors, *Proceedings of DaWaK-00, 2nd International Conference on Data Warehousing and Knowledge Discovery*, pages 409–418, London, UK, 2000. Springer Verlag, Heidelberg, DE.

[16] Thomas G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science (LNCS)*, 1857:1–15, 2000.

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

[18] Ming Dong and Ravi Kothari. Feature subset selection using a new definition of classifiability. *Pattern Recognition Letters*, 24(9-10):1215–1225, 2003.

[19] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification and Scene Analysis, 2nd Ed.* Wiley-Interscience, October 2000.

[20] Susan Dumais and Hao Chen. Hierarchical classification of web content. In *Proc. ACM SIGIR*, pages 256 – 263, 2000.

[21] Frank Eibe and Stefan Kramer. Ensembles of nested dichotomies for multi-class problems. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, pages 305–312, 2004.

[22] R. Erenshteyn and P. Laskov. A multi-stage approach to fingerspelling and gesture recognition. In *Proceedings of the Workshop on the Integration of Gesture in Language and Speech*, pages 185–194, 1996.

[23] D.H. Foley and J.W. Sammon. An optimal set of discriminant vectors. *IEEE Transactions on Computing*, 24:271–278, 1975.

[24] D Fradkin and I Muchnik. A study of k-means algorithm for improving classification accuracy of multi-class svm. In *DIMACS Technical Report*, 2004.

[25] Freund. An adaptive version of the boost by majority algorithm. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*, 1999.

[26] Y. Freund and R. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, 1999.

[27] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1996.

[28] Ingo Frommholz. Categorizing Web documents in hierarchical catalogues. In *Proceedings of ECIR-01, 23rd European Colloquium on Information Retrieval Research*, Darmstadt, DE, 2001.

[29] Johannes Fürnkranz. Round robin rule learning. In C. E. Brodley and A. P. Danyluk, editors, *Proceedings of the 18th International Conference on Machine Learning (ICML-01)*, pages 146–153, Williamstown, MA, 2001. Morgan Kaufmann Publishers.

[30] Saul B. Gelfand, C. S. Ravishankar, and Edward J. Delp. An iterative growing and pruning algorithm for classification tree design. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 13(2):163–174, February 1991.

[31] Zoubin Ghahramani and Hyun-Chul Kim. Bayesian classifier combination. *Technical Report, University College London*, 2003.

[32] Shantanu Godbole, Sunita Sarawagi, and Soumen Chakrabarti. Scaling multi-class support vector machines using inter-class confusion. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 513–518, 2002.

[33] M. Golea and M. Marchand. A growth algorithm for neural network decision trees. *Euro- Physics Letters*, 12(3):205–210, June 1990.

[34] S. Gutta and H. Wechsler. Face recognition using hybrid classifier systems. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1017–1022, 1996.

[35] W. Hanisch. Design and optimization of a hierarchical classifier. *Journal of new Generation Computer Systems*, 3(2):159–173, 1990.

[36] D. Heath, S. Kasif, and S. Salzberg. Learning oblique decision trees. *IJCAI*, 160:1002–1007, 1993.

[37] Seong-Whan Lee Hee-Seon Park, Hee-Heon Song. A self-organizing hierarchical classifier for multi-lingual large-set oriental character recognition. *IJPRAI*, 12(2):191 – 208, 1998.

[38] B. Heisele, T. Serre, M. Pontil, T. Vetter, and T. Poggio. Categorization by learning and combining object parts. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 1239–1245, Cambridge, MA, 2002. MIT Press.

[39] Bernd Heisele, Thomas Serre, Sam Prentice, and Tomaso Poggio. Hierarchical classification andfeature reduction for fast face detection with support vector machines. *Pattern Recognition*, 36:2007 – 2017, 2003.

[40] S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases. 1998,
(*http://www.ics.uci.edu/∼mlearn/mlrepository.html*).

[41] C. W. Hsu and C. J. Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13:415–425, 2002.

[42] L. Hyafil and R. L Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5:15–17, 1976.

[43] I.K.Sethi. Machine recognition of constrained handprinted Devanagari. In *Pattern Recognition*, volume 9, pages 69–75, 1977.

[44] Christopher J.C.Burges. A tutorial on support vector machines for pattern recognition. In *Data Mining and Knowledge Discovery*, 1998.

[45] Thorsten Joachims. Making large-scale svm learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT-Press, 1999.

[46] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *Proc. 16th International Conference on Machine Learning*, pages 200–209. Morgan Kaufmann, San Francisco, CA, 1999.

[47] Tapas Kanungo et al. A statistical, nonparametric methodology for document degradation model validation. *IEEE Tran. PAMI*, pages 1209–1223, 2000.

[48] G. Karypis. Cluto : A clustering toolkit. Technical Report TR-02-017, Department of Computer Science, University of Minnesota, Minneapolis, 2003.

[49] G. Karypis, E.H. Han, and V. Kumar. Multilevel refinement for hierarchical clustering. Technical Report TR-99-020, Department of Computer Science, University of Minnesota, Minneapolis, 1999.

[50] G. Karypis and V. Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1), 1999.

[51] J Kittler. *Handbook of Pattern Recognition and Image Processing.* Academic Press, New York, 1986.

[52] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In Douglas H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 170–178, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.

[53] Shailesh Kumar and Joydeep Ghosh. GAMLS: A generalized framework for associative modular learning systems. *Application and Science of Computational Intelligence II, SPIE*, 3722:24–35, April 1999.

[54] Shailesh Kumar, Joydeep Ghosh, and Melba Crawford. A hierarchical multiclassifier system for hyperspectral data analysis. *Lecture Notes in Computer Science*, 1857:270–278, 2000.

[55] M. N. S. S. K. Pavan Kumar and C. V. Jawahar. Design of Hierarchical Classifier with Hybrid Architectures. In *Proc. of First International Conference on Pattern Recognition and Machine Intelligence(PReMI)*, page (in Press), Dec 2005.

[56] M. N. S. S. K. Pavan Kumar and C. V. Jawahar. Configurable Hybrid Architectures for Character Recognition Applications. In *International Conference on Document Analysis and Recognition (ICDAR)*, page (in Press), Sep 2005.

[57] M N S S K Pavan Kumar and C. V. Jawahar. On improving design of multiclass clasifiers. In *Proceedings of the 5th International Conference on Advances in Pattern Recognition*, pages 109–112, 2003.

[58] S. S. Marwah, S. K. Mullick, and R. M. K. Sinha. Recognition of Devanagari characters using a hierarchial binary decision tree classifier. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 414–420, October 1984.

[59] Andrew K. McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In Jude W. Shavlik, editor, *Proceedings of ICML-98, 15th International Conference on Machine Learning*, pages 359–367, Madison, US, 1998. Morgan Kaufmann Publishers, San Francisco, US.

[60] Tom M. Mitchell. *Machine Learning.* McGraw Hill, U.S.A, 1997.

[61] Sreerama K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4):345 – 389, December 1998.

[62] Sreerama K. Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

[63] Sreerama K. Murthy, Simon Kasif, Steven Salzberg, and Richard Beigel. OC1: A randomized induction of oblique decision trees. In *National Conference on Artificial Intelligence*, pages 322–327, 1993.

[64] M.Vidyasagar. *A Theory of Learning and Generalization*. Springer-Verlag,New York, 1997.

[65] Shaoning Pang, Daijin Kim, and Sung Yang Bang. Face membership authentication using svm classification tree generated by membership-based lle data partition. *IEEE Trans Neural Networks*, 16(2):436–446, March 2005.

[66] Shaoning Pang, Daijin Kim, and Sung Yang Bang. Face membership authentication using svm classification tree generated by membership-based lle data partition. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 16(2), March 2005.

[67] John C. Platt, Nello Cristianini, and John Shawe-Taylor. Large margin DAGs for multi-class classification. In *Advances in NIPS-12*, pages 547–553, 2000.

[68] J. R. Quinlan. Learning decision tree classifiers. *ACM Computing Surveys (CSUR)*, 28(1):71–72, March 1996.

[69] John Ross Quinlan and Ronald L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80(3):227–248, March 1989.

[70] Suju Rajan and Joydeep Ghosh. An empirical comparison of hierarchical vs. two-level approaches to multiclass problems. In *Proceedings of the 5th International Workshop on Multiple Classifier Systems*, pages 283 – 292, 2004.

[71] Fabio Roli, Giorgio Giacinto, and Gianni Vernazza. Methods for designing multiple classifier systems. *Lecture Notes in Computer Science*, 2096:78–87, 2001.

[72] Miguel E. Ruiz and Padmini Srinivasan. Hierarchical text categorization using neural networks. *Inf. Retr.*, 5(1):87–118, January 2002.

[73] S.R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man and Cybernetics*, 21(3):660–674, May 1991.

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

[75] Robert E. Schapire. A brief introduction to boosting. In *IJCAI*, pages 1401–1406, 1999.

[76] F. Schwenker. Hierarchical support vector machines for multi-class patternrecognition. In *Proceedings of the Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies*, volume 2, pages 561–565, 2000.

[77] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(8):888–905, 2000.

[78] J.A. Sirat and J.-P. Nadal. Neural trees: A new tool for classification. *Computation in Neural Systems*, 1(4):423–438, October 1990.

[79] Aixin Sun and Ee-Peng Lim. Hierarchical text classification and evaluation. In *ICDM*, pages 521–528, 2001.

[80] Paul E. Utgoff. Perceptron trees: A case study in hybrid concept representations. *Connection Science*, 1(4):377–391, 1989.

[81] Vladimir Vapnik. *The Nature of Statistical Learning Theory.* New York: Springer, 1995.

[82] Volkan Vural and Jennifer G. Dy. A hierarchical method for multi-class support vector machines. In *Proceedings of the 21st ICML*, pages 831–838, July 2004.

[83] Ke Wang, Senquiang Zhou, and Shiang Chen Liew. Building hierarchical classifiers using class proximity. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *Proceedings of VLDB-99, 25th International Conference on Very Large Data Bases*, pages 363–374, Edinburgh, UK, 1999. Morgan Kaufmann Publishers, San Francisco, US.

[84] Qing Ren Wang and C. Y. Suen. Analysis and design of a decision tree based on entropy reduction and its application to large character set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:406–417, 1984.

[85] J. Weston and C. Watkins. Multi-class support vector machines. Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London, Egham, TW20 0EX, UK, 1998.

[86] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 15(11):1101–1113, 1993.

[87] Olcay Taner Yildiz and Etbern Aplaydin. Omnivariate decision trees. *IEEE Transactions on Neural Networks*, 12(6):1539–1546, November 2001.

[88] Xiaoqing Ding Yong Ma. Face detection based on hierarchical support vector machines. In *16th International Conference on Pattern Recognition*, pages 102–106, 2002.

[89] Gary Geunbae Lee Yongwook Yoon, Changki Lee. Systematic construction of hierarchical classifier in svm-based text categorization. *Lecture Notes in Computer Science*, 3248:616–625, January 2005.

[90] H. Yu, J. Yang, and J. Han. Classifying large data sets using svm with hierarchical clusters. In *Proc. 2003 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'03)*, August 2003.

[91] H. Yu, J. Yang, and J. Han. Classifying large data sets using svm with hierarchical clusters. In *Proc. 2003 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'03)*, pages 306–315, 2003.

[92] H. Zha, X. He, C. Ding, H. Simon, and M. Gu. Bipartite graph partitioning and data clustering. In *CIKM*, 2001.