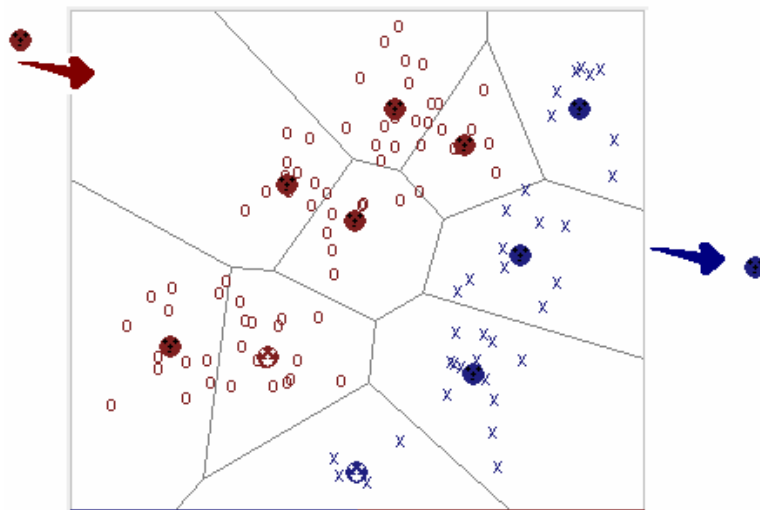


“Dynamic GRLVQ” Algorithm

An application to cancer prediction and Bioinformatics

**University of Osnabrück, Germany
Department of Mathematics and Computer Science**

**University of Twente, Netherlands
Department of Computer Science**



**Thesis by
Gulraj Singh**

**In fulfilment of the requirements for the degree of
Master of Science in Information Engineering**

**Osnabrück, Germany, 2004
Submitted on Tuesday, 23 November 2004**

Referees:

- **Prof. Barbara Hammer, Theoretical Computer Science, Technical University of Clausthal, Germany.**
- **Dr. Thomas Villmann, Research Group “Computational Intelligence”, University of Leipzig, Germany**

Abstract

This work is about machine learning methods for problems in information retrieval and bioinformatics, in particular cancer prediction. This aims at predicting whether a probe is inflicted by cancer or not and, in the former case, which type of cancer, based on given spectrometric data.

For machine learning, exemplary classified examples for different types of cancer (prostate and ovarian) are available such that supervised machine learning can be used for this task. In this thesis, several machine learning methods will be discussed and one specific method, so-called generalized relevance learning vector quantization (GRLVQ) which is a very intuitive prototype based classification method, will be implemented.

One problem of cancer prediction is given by the fact that data are usually very high dimensional such that the curse of dimensionality applies. In addition, one is interested in a semantic interpretation of the results in form of biomarkers, i.e. specific dimensions of the spectral information which are of particular relevance for the classification and which correspond to the fingerprint of products within the serum which are due to the cancer. Apart from their significance for the classification, these biomarkers might indicate a direction for possible therapies. GRLVQ can deal with these problems because of an adaptive relevance weighting scheme which is included in the algorithm. In the thesis, this fact will be demonstrated and alternative possibilities to reduce the data dimensionality and to extract significant dimensions will be discussed for convenience.

When using GRLVQ it is necessary to priorly decide about the final model complexity as for many learning algorithms. Since the model complexity is not known in our application examples, we avoid this problem by introducing a dynamic version of GRLVQ, Dynamic GRLVQ (DGRLVQ), which adapts the model complexity to the given problem during training. The efficiency and accuracy of the method will be demonstrated in several examples.

Speaking from an abstract point of view, the thesis deals with automatic classification and data mining tasks in the presence of very high dimensional data. Thereby, automatic adaptation of the model and the model complexity are implemented and applied to recent problems in bioinformatics.

Keywords: Dynamic generalization relevance learning vector quantization DGRLVQ, LVQ, GRLVQ, clustering, classification, information retrieval, machine learning, adaptive metric, cancer, proteomic patterns

Acknowledgment

I am indebted to all who encouraged me directly or indirectly to work over DGRLVQ, especially **Dr. Barbara Hammer***. She is one who detailed me in depth working of GRLVQ and may have spent hours to generate research material for me. I am very thankful to her for all those helpful advices and suggestions that motivated me on DGRLVQ algorithm.

I thank **Dr. Thomas Villmann*** for assessing this work and providing test materials. He always supported this work with great interest and enthusiasm.

Although I had very small and precise discussions in cafeteria with **Dr. Roland Schwänzl***, he always left me thinking interesting ideas. His guidance in mathematical clustering and classifications was a great help.

I am thankful to **Dr. Marc Strickert***. His constant guidance and patience helped me a lot in the development on java package for DGRLVQ. Those hours' long discussions with him over algorithm design and implementation were great success to achieve the target results.

Prof. Dr. Barbara Hammer*: Prof. Barbara Hammer, Theoretical Computer Science, Technical University of Clausthal, Germany.

Prof. Dr. Roland Schwänzl*: Department of Mathematics, University Osnabrück, Germany.

Prof. Dr. Thomas Villmann*: Research Group "Computational Intelligence", University of Leipzig, Germany

Dr. Marc Strickert*: Pattern recognition group, Cytogenetics department, IPK Gatersleben, Germany.

Table of Contents

Abstract

Acknowledgments

1. Introduction

2. Pattern Recognition Systems

2.1 Learning modes of Pattern Recognition System

 2.1.1 Supervised learning

 2.1.2 Unsupervised learning

2.2 Learning Models

 2.2.1 ANNs vs. statistical methods

 2.2.2 Artificial neural network

 2.2.3 Decision trees

 2.2.4 Support vector machines

 2.2.5 K-NN

 2.2.6 VQ

 2.2.7 K-means

 2.2.8 SOM

 2.2.9 LVQ

3. Data pre processing & Feature Extraction

3.1 Simplest Approach: Normalization, Standardization and Rescaling

3.2 Complex Approach: Dimensionality reduction and feature extraction

 3.2.1 Dimensionality reduction with Principle Components Analysis (PCA)

 3.2.2 Feature Extraction

3.3 Generalization

 3.3.1 Split-sample or hold-out validation

 3.3.2 Cross-validation

 3.3.3 Bootstrapping

4. Performance Measures

 4.1 Percent Correct Method

 4.2 Receiver Operating Curves (ROC)

 4.3 Specificity and Sensitivity

5. Pattern Recognition with Dynamic-GRLVQ Algorithm

 5.1 Basic DGRLVQ

 5.2 Pseudocode

6. Applications and Experiments

 6.1 Artificial Geographical satellite data

 6.2 Iris

 6.3 Mushroom

 6.4 Ovarian Cancer

 6.5 Prostate Cancer

Conclusions and discussion

References

1 Introduction

An exponential growth of world economy and industry brought with it a huge bulk of data. Images, DNA sequences, documents and medical reports etc are some of the examples. Along with this data comes the need to develop efficient methods for automatic data processing. This includes several different tasks according to the large variety of different application areas: data mining, information extraction, function approximation, dimensionality reduction or classification, to name just few.

This thesis will mainly focus on supervised classification tasks, i.e. to classify data into sets of different classes that share the common properties so as to gain information which is used for a number of industrial applications. In other words, we need efficient techniques for assigning data to their respective classes automatically with as little human intervention as possible.

Classification is the task of classifying the members of a given set of objects into groups on the basis of whether they have some common properties or not. Some typical classification tasks are 1) medical testing to determine if a patient has certain disease or not (the classification property is the disease) 2) quality control in factories; i.e. deciding if a new product is good enough to be sold, or if it should be discarded (the classification property is being good enough) 3) deciding whether a page or an article should be in the result set of a search or not (the classification property is the relevance of the article - typically the presence of a certain word in it)

Sometimes, classification tasks are trivial. Given 100 balls, some of them red and some blue, a human with normal color vision can easily separate them into red ones and blue ones. However, some tasks, like those in practical medicine, and those interesting from the computer science point-of-view, are far from trivial, and produce also faulty results. Moreover, data that has to be classified come in many different shapes and sizes and assigning of data into different classes to get information is not always an easy task because data can be very complex, noisy, redundant, inconclusive, missing and high dimensional. Under such circumstances, to evaluate results and to retrieve information from data has become a challenge work within research communities. So a need to develop an efficient automatic classification procedures have become prominent.

To make things more clear let us consider an example. In this thesis, the focus of the application will lay on the automatic classification of medical data, in particular **Cancer Prediction**. One popular method in cancer research to detect disease is the generation of Proteomic Patterns from human serum by mass spectroscopy and to compare this pattern

against a healthy or diseased Pattern. The generated pattern is classified as Cancer if it has the similar properties as cancer pattern or else it is classified as Non-Cancer. Here, we are dealing with classifying Proteomic patterns in the category of cancer and non-cancer. An expert human can do this kind of classification manually by recognizing some of the features of cancer pattern from those not present in non-cancer pattern. The question arises is how this expert can make such kind of decisions? The simple answer is that this expert has already learned from a number of different pattern examples (cancer and non-cancer patterns) and has the ability to recognize now the difference between cancer and non-cancer pattern just by looking at them. Cancerous cells yield to specific peptides within the human serum which are different compared to healthy matter. These charged proteomic pattern often manifests itself by characteristic peaks indicating biomarkers within the spectrum. But for a given large set of patterns for classification, manual classification is not the ideal way as it consumes lots of both money and time. In addition, if biomarkers are lacking and only peak groups at a large range of the spectrum allow an appropriate diagnosis, the problem no longer is solved by manual investigation at all. Thus automatic pattern recognition tools for cancer classification are highly desirable.

Within this thesis, automatic learning will take place by the methods of **artificial intelligence** which mimic human intelligence to achieve high level autonomous intelligent systems. Humans have a natural mental capability to reason. They learn by a process of acquiring modifications in existing knowledge through experience and hence are intelligent. So this intelligent human being developed Machines (for e.g. Computers terminals, television, mobile phones and washing machines etc) to amplify and to replace human effort to accomplish difficult tasks. Over many years, humans could maintain only a master-slave relationship with machines because machines were able to understand only a discrete set of commands. But the scenario changed dramatically with the evolution of artificial intelligence. To do automate classification and pattern recognition via machines, the subject of Machine learning came in picture. These days, machines are no more dumb terminals but machine intelligence allows computers to learn and take decisions in the same way as humans do. Machine Pattern reorganization has a wide spectrum of applications including medical diagnosis to predict disease stage, detecting credit card fraud, stock market analysis, classifying DNA sequences, speech, face, fingerprint and handwriting recognition, game playing, robot locomotion, stock markets and industrial economics etc.

To apply machine learning techniques successfully to large-scale real-life problems, still several issues have to be taken into account. As we already stated that classification data can be noisy and redundant so, **data preprocessing** should be considered before data are fed to any machine learning algorithm. Preprocessing of data includes data cleaning, conversion, transformation and well formation that enhance the efficiency and reliability of classifier algorithm.

Often, we still need to face the **curse of dimensionality and best feature selection** of data. To explain it briefly, if we think about how we classify common everyday objects such as animals and cars, we are using **features** of those objects to classify them correctly. For every object of the kind “Animal” we have features like legs and color. For every object of the kind “Car” we have features like wheels and color. This is a 2- dimensional classification problem where we need to have some knowledge of what features make good predictors of class membership for the classes we are trying to distinguish. Having legs or wheels distinguishes all objects of the kind “Animal” from objects of the kind “Car” whereas we cannot guess anything taking only Color feature into consideration because both animals and cars can have the same color. This means, we can just neglect the color feature for our classification problem and we can still predict accurately about class membership of the object based only on one feature. Often, several hundred and thousands of features are available for a concrete learning problem and no prior information is available which features are best suited for the current task. In addition, only few training examples might be available such that the data space is only sparsely covered.

In the example above, we have selected the appropriate feature that resulted in dimensionally reduction (from 2D to 1D) of data and we can still do a good job of classification. Depending on the classification task we are facing, different features or sets of features may be important, and knowing how we arrive at our knowledge of which features are useful to which task is essential. An inbuilt capability of the classifier to **select relevant features and dimensionality reduction** is important in such situations.

Different classification algorithms perform differently on a given set of conditions. Some are fast but cannot handle complex data with high dimensions and others are slow but achieve a good accuracy also in high dimensions. There exist both good and bad classifiers and **measuring how well a classifier is performing** is not straightforward. One way is to find simply what percentage of correct classifications is possible and compare it with the specification that was established beforehand. Another way is to conclude how well a classifier performs with respect to some specified tolerance. Benchmarking of classifier is important as it helps us to select a classifier algorithm for certain situations under certain conditions.

Classification in general is one of the problems studied in computer science in order to automatically learn classification systems. There exist a number of efficient machine learning algorithms for classification and pattern reorganization problems like decision trees, neural networks, support vector machine and prototype based classifiers but the presence of high dimensionality in data makes the job complicated. Reducing dimensionality of data (with PCA etc) becomes important before data is fed to these algorithms but this also reduces the information that is contained in the data. On the other hand if no dimensionality reduction is performed, a complex solution model is generated by these algorithms. In bioinformatics, we

generally have high dimensional data and the problem is double folded when the number of available examples is small which is true in many cases. Another problem is that the binary classifications are often not sufficient; we need algorithms which can classify data into a number of classes parallel and fast.

In this thesis, we introduce the Dynamic-GRLVQ algorithm that is q prototype based architecture with dynamically increasing and shrinking network sizes. We investigate the benefits of this new architecture over other machine learning algorithms in the presence of high dimensional multiclass data and demonstrate how the Dynamic-GRLVQ algorithm learns fast even in the presence of very few data examples in several medical tasks. Further we discuss our results in details and give suggestions for future research.

2 Pattern Recognition System

Pattern recognition is the research area that studies the operation and design of systems that recognize patterns in data. The detection of underlying patterns is called pattern recognition [9]. Broadly speaking, pattern reorganization involves assignment of input pattern in meaningful categories. It involves extraction of significant attributes (feature extraction) of the pattern from the background of irrelevant details. It encloses sub disciplines like discriminant analysis, feature extraction, error estimation, cluster analysis (together sometimes called statistical pattern recognition), grammatical inference and parsing (sometimes called syntactical pattern recognition). Important application areas are image analysis, character recognition, speech analysis, man and machine diagnostics, person identification and industrial inspection.

To determine the class of a pattern is one of the major jobs of pattern recognition. A raw pattern vector is given to a pattern recognition system and selecting relevant attributes from this pattern vector is typically an essential part of such a system. These relevant attributes clearly represent the underlying structure of the pattern. Pattern recognition system generally contains several stages:

1. Preprocessing and conversion of data objects to pattern vectors, Feature extraction and abstraction of high level information about individual patterns to facilitate recognition.
2. Classifier: learn over training pattern set, construct a learning model and if given a new pattern that didn't belong to the training set then to identify the pattern class (category) to which the pattern belongs.
3. Evaluation and postprocessing.

There exist a number of efficient learning algorithms to construct an efficient classifier. The selection of learning algorithm is highly based upon the kind of problem we face. Moreover we can also construct a hybrid classifier to our ease.

Pattern classification systems are commonly categorized in the terms of their corresponding training algorithms. There exists two major possibilities **1) Supervised classification**; where a given pattern has to be classified as a member of an already known/defined class **2) Unsupervised classification**; where a pattern needs to be assigned to a so far unknown class pattern. Both of these classifications can be achieved based upon the kind of learning mode implemented by a pattern recognition system.

2.1 Learning modes of Pattern Recognition System

Learning is the general process by which we gain knowledge of which features matter in a given discrimination task. For those of us who are parents, one example of this type of learning (feature selection) involves teaching our children about types of animals. We (endlessly) point to animals and say words like dog or cat or horse. We don't generally give our children a feature list that a biologist might use to define *Canis familiaris* or *Felis catus* [19]. Instead, we present examples and expect our children to figure out for themselves what the important features are. And when they make a correct guess about an animal (a correct classification or prediction), we give copious amounts of positive feedback.

This procedure is called **supervised learning**. We present our children or our computer programs with examples and tell them what category each example belongs to, so they learn under our supervision. This is in contrast to **unsupervised learning**. In unsupervised learning objects are grouped together based on perceptions of similarity (or more properly, relative lack of difference) without anything more to go on. While unsupervised learning is indispensable, supervised learning has a substantial advantage over unsupervised learning.

In particular, supervised learning allows us to take advantage of our own knowledge about the classification problem we are trying to solve. Instead of just letting the algorithm work out for itself what the classes should be, we can tell it what we know about the classes: how many there are and what examples of each one look like. The supervised learning algorithm's job is then to find the features in the examples that are most useful in predicting the classes.

The standard formulation of such a learning task is as a classification problem. The learner is required to learn (to approximate the behaviour of) a function which maps a vector $V = (x_1, x_2, \dots, x_n)$ into one of several classes by looking at several input-output examples of the function.

2.1.1 Supervised learning

The algorithm generates a function that maps inputs to desired outputs. Supervised learning is a machine learning technique for creating a function from training data. The training data consists of pairs of input objects (typically vectors), and desired outputs. The output of the function can be a continuous value (called regression), or can predict a class label of the input object (called classification). The task of the supervised learner is to predict the value of the function for any valid input object after having seen only a small number of training examples (i.e. pairs of input and target output). To achieve this, the learner has to generalize from the presented data to unseen situations in a "reasonable" way.

In order to solve a given problem of supervised learning (e.g. learning to recognize handwriting) one has to consider various steps:

Determine the type of training examples. Before doing anything else, the engineer should decide what kind of data is to be used as an example. For instance, this might be a single handwritten character, an entire handwritten word, or an entire line of handwriting.

Gathering a training set. The training set needs to be characteristic of the real-world use of the function. Thus, a set of input objects is gathered and corresponding outputs are also gathered, either from human experts or from measurements.

Determine the input feature representation of the learned function. The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a number of features that are descriptive of the object. The number of features should not be too large, because of the curse of dimensionality; but should be large enough to accurately predict the output.

Determine the structure of the learned function and corresponding learning algorithm. For example, the engineer may choose to use neural networks or decision trees.

Complete the design. The engineer then runs the learning algorithm on the gathered training set. Parameters of the learning algorithm may be adjusted by optimizing performance on a subset (called a validation set) of the training set, or via cross-validation. After parameter adjustment and learning, the performance of the algorithm may be measured on a test set that is separate from the training set.

2.1.2 Unsupervised learning

The algorithm generates a model for a set of inputs. Unsupervised learning is a method of machine learning where a model is fit to observations. It is distinguished from supervised learning by the fact that there is no a priori output. In unsupervised learning, a data set of input objects is gathered. Unsupervised learning then typically treats input objects as a set of random variables. A joint density model is then built for the data set.

Unsupervised learning can be used in conjunction with Bayesian inference to produce conditional probabilities (i.e., supervised learning) for any of the random variables given the others. Unsupervised learning is also useful for data compression: fundamentally, all data compression algorithms either explicitly or implicitly rely on a probability distribution over a set of inputs. Another form of unsupervised learning is clustering, which is sometimes not probabilistic.

2.2 Learning Models

There exists a verity of different learning models. Here, we give a start overview about different models. Within this thesis, the focus will lie on a specific neural classification method. Neural models and statistical classifiers constitute major approaches within pattern recognition.

2.2.1 ANNs vs. statistical methods

There is considerable overlap between the fields of neural networks and statistics. Statistics is concerned with data analysis. In neural network terminology, statistical inference means learning to generalize from noisy data. Some neural networks are not concerned with data analysis (e.g., those intended to model biological systems) and therefore have little to do with statistics. But most neural networks that can learn to generalize effectively from noisy data are similar or identical to statistical methods.

While neural nets are often defined in terms of their algorithms or implementations, statistical methods are usually defined in terms of their results. It is sometimes claimed that neural networks, unlike statistical models, require no distributional assumptions. In fact, neural networks involve exactly the same sort of distributional assumptions as statistical models but statisticians study the consequences and importance of these assumptions while many neural net workers ignore them. For example, least-squares training methods are widely used by statisticians and neural net workers. Statisticians realize that least-squares training involves implicit distributional assumptions in that least-squares estimates have certain optimality properties for noise that is normally distributed with equal variance for all training cases and that is independent between different cases. These optimality properties are consequences of the fact that least-squares estimation is maximum likelihood under those conditions. Similarly, cross-entropy is maximum likelihood for noise with a Bernoulli distribution. If you study the distributional assumptions, then you can recognize and deal with violations of the assumptions. For example, if you have normally distributed noise but some training cases have greater noise variance than others, then you may be able to use weighted least squares instead of ordinary least squares to obtain more efficient estimates.

We shall concentrate mostly on Pattern recognition systems based upon artificial neural network (ANN) techniques, especially Prototype based learning methods, and in particular, we propose the Dynamic-GRLVQ algorithm and construct a number of applications using it.

2.2.2 Artificial neural network

Most artificial neural networks (ANNs) are motivated by biological neural networks. Much of the inspiration for the field of ANNs came from the desire to produce artificial systems

capable of sophisticated, perhaps "intelligent", computations similar to those that the human brain routinely performs, and thereby possibly to enhance our understanding of the human brain. Biological neural networks are much more complicated than the mathematical models we use for ANNs.

An ANN is a network of many simple processors ("units"), each possibly having a small amount of local memory. The units are connected by communication channels ("connections") which usually carry numeric (as opposed to symbolic) data, encoded by any of various means. The units operate only on their local data and on the inputs they receive via the connections.

NNs "learn" from examples, as children learn to distinguish dogs from cats based on examples of dogs and cats. If trained carefully, NNs may exhibit some capability for generalization beyond the training data, that is, to produce approximately correct results for new cases that were not used for training. NNs normally have great potential for parallelism, since the computations of the components are largely independent of each other. An ANN resembles the brain in two respects:

1. Knowledge is acquired by the network through a learning process.
2. Interneuron connection strengths known as synaptic weights are used to store the knowledge.

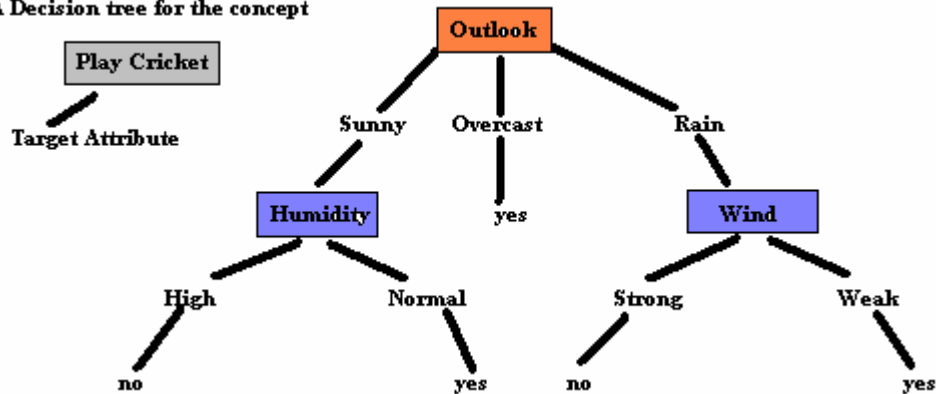
In principle, NNs can compute any computable function, i.e., they can do everything a normal digital computer can do. In practice, NNs are especially useful for classification and function approximation/mapping problems which are tolerant of some imprecision, which have lots of training data available, but to which hard and fast rules (such as those that might be used in an expert system) cannot easily be applied. There are many kinds of NNs by now both for supervised and unsupervised tasks.

2.2.3 Decision trees

In decision theory (for example risk management), a decision tree is a graph of decisions and their possible consequences, (including resource costs and risks) used to create a plan to reach a goal. Decision trees are constructed in order to help with making decisions.

In machine learning, a decision tree is a predictive model; that is, a mapping of observations about an item to conclusions about the item's target value. Each inner node corresponds to a variable; an arc to a child represents a possible value of that variable. A leaf represents the predicted value of a target variable given the values of the variables represented by the path from the root. The machine learning technique for inducing a decision tree from data is called decision tree learning, or Decision Trees. Given below is a small toy example:

A Decision tree for the concept



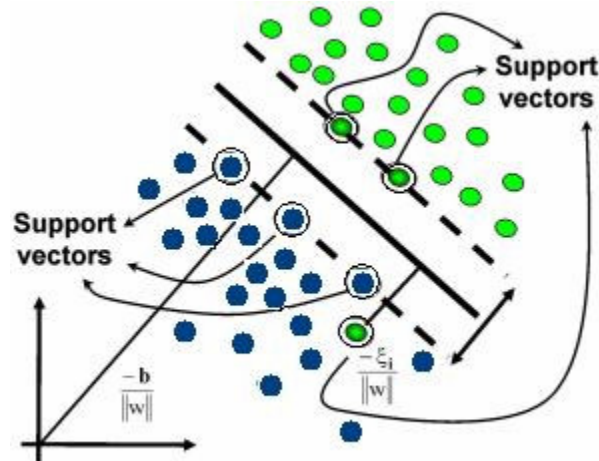
Decision tree learning is also a common method used in data mining. Here, a decision tree describes a tree structure wherein leaves represent classifications and branches represent conjunctions of features that lead to those classifications. A decision tree can be learned by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner. The recursion is completed when splitting is either non-feasible, or a singular classification can be applied to each element of the derived subset. Popular learning models include ID3 and C4.5

2.2.4 Support vector machines

Support vector machines (SVMs) are applicable to both classification and regression tasks. The SVM basically consists of a simple linear classifier with fixed Nonlinear preprocessing of the data. The generalization ability of the classifier to new data is very good because the algorithm searches for a very robust solution: the classifier with maximum margin. In addition, the nonlinear preprocessing is computed via so-called kernels, i.e. a function which allows computing the dot product in the high-dimensional feature space efficiently.

The basic classification SVM creates a maximum-margin hyper plane that lies in a transformed input space. Given training examples labelled either "yes" or "no", a maximum-margin hyperplane splits the "yes" and "no" training examples, such that the distance from the closest examples (the margin) to the hyperplane is maximized. The use of the maximum-margin hyperplane is motivated by statistical learning theory, which provides a probabilistic test error bound which is minimized when the margin is maximized. The parameters of the maximum-margin hyperplane are derived by solving a quadratic programming (QP) optimization problem. There exist several specialized algorithms for quickly solving the QP problem that arises from SVMs.

Training SVM is some easy form of constraint quadratic optimization, and various different optimization schemes have been proposed. The trained SVM can be expressed in terms of so-called support vectors; these are vectors at the classification boundary. Thus solutions will be expressed in terms of extreme data points from the training set.



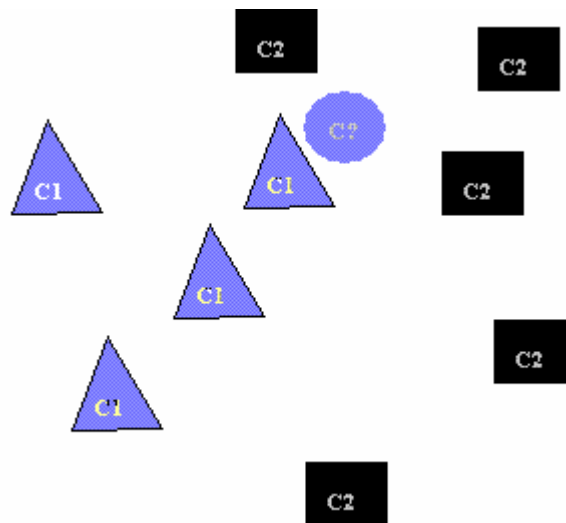
The original optimal hyperplane algorithm proposed by Vladimir Vapnik was a linear classifier, Bernhard Boser, Isabelle Guyon and Vapnik suggested applying the kernel trick (originally proposed by Aizerman) to maximum-margin hyperplanes. The resulting algorithm is formally similar, except that every dot product is replaced by a non-linear function. This causes the linear algorithm to operate in a different space. Using the kernel trick makes the maximum margin hyperplane be fit in a feature space. The feature space is a non-linear map from the original input space, usually of much higher dimensionality than the original input space. In this way, non-linear classifiers can be created. If the kernel used is a radial basis function, the corresponding feature space is a Hilbert space of infinite dimension. Maximum margin classifiers are well regularized, so the infinite dimension does not spoil the results.

The model produced by Support Vector Classification (as described above) only depends on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by Support Vector Regression only depends on a subset of the training data, because the cost function for building the model ignores any training data that is close (within a threshold ϵ) to the model prediction.

SVM has very good generalization ability, quadratic time training, flexibility due to kernels and is a very popular method for classification problems but the representation of the classifier in terms of extreme values is somewhat unnatural and the size of the classifier scales with the nature of training data.

2.2.5 K-NN

K-NEAREST NEIGHBOR is a simple algorithm that stores all available examples and classifies new instances of the example language based on a similarity measure. This approach is suited for function approximation as well. Instead of assigning the most frequent classification among the k examples most similar to an instance e , an average of the function values of the k examples is calculated as the prediction for the function value e .



1-NN

A variant of this approach calculates a weighted average of the nearest neighbours. Given a specific instance e that shall be classified, the weight of an example increases with increasing similarity to e .

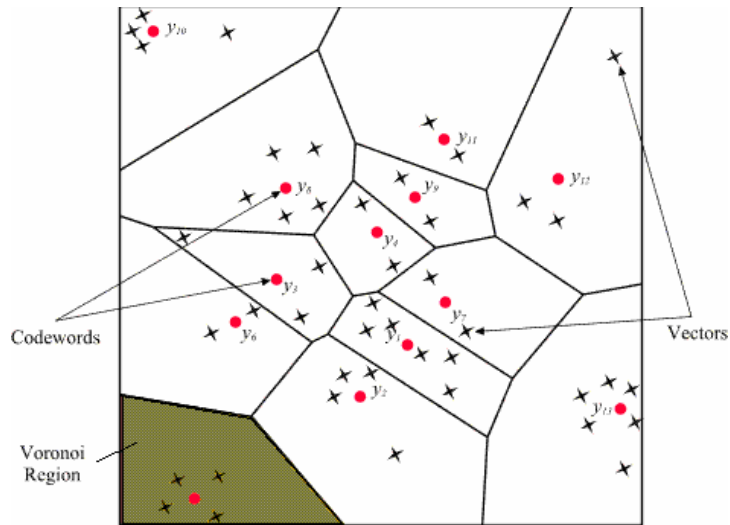
A major problem of the simple approach of k -NEAREST NEIGHBOR is that the vector distance will not necessarily be suited for finding intuitively similar examples, especially if irrelevant attributes are present. In addition, this approach is quite costly since all training data are stored.

2.2.6 VQ

Unlike the previous algorithms, Vector Quantization constitutes an unsupervised learning approach. VQ is a competitive network that can be viewed as unsupervised density estimators or autoassociators, closely related to k -means cluster analysis. Each competitive unit corresponds to a cluster, the centre of which is called a "codebook vector". Kohonen's learning law is an on-line algorithm that finds the codebook vector closest to each training

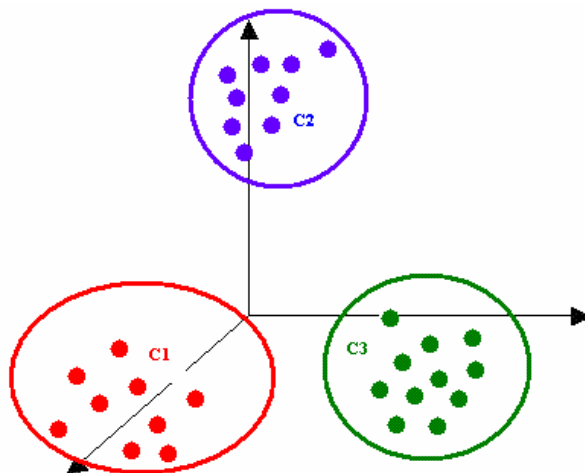
case and moves the "winning" codebook vector closer to the training case. The codebook vector is moved a certain proportion of the distance between it and the training case, the proportion being specified by the learning rate, that is:

$$\text{new_codebook} = \text{old_codebook} * (1 - \text{learning_rate}) + \text{data} * \text{learning_rate}$$



2.2.7 K-means

This algorithm is essentially the same as Kohonen's learning law except that the learning rate is the reciprocal of the number of cases that have been assigned to the winning cluster.



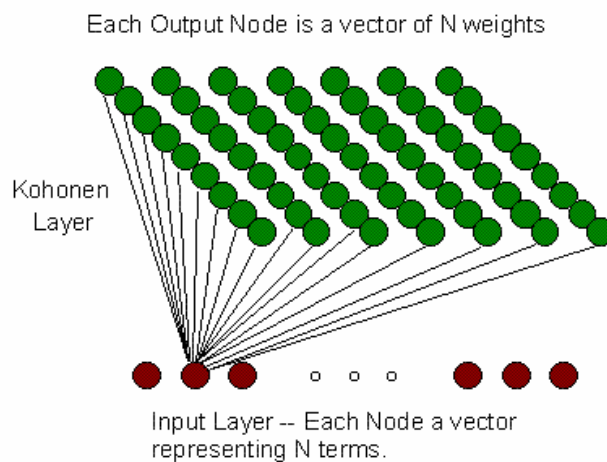
Suppose that when processing a given training case, N cases have been previously assigned to the winning codebook vector. Then the codebook vector is updated as:

$$\text{new_codebook} = \text{old_codebook} * N / (N+1) + \text{data} * 1 / (N+1)$$

This reduction of the learning rate makes each codebook vector the mean of all cases assigned to its cluster and guarantees convergence of the algorithm to an optimum value of the error function (the sum of squared Euclidean distances between cases and codebook vectors) as the number of training cases goes to infinity.

2.2.8 SOM

Self-Organizing Map--competitive networks that provide a "topological" mapping from the input space to the clusters (Kohonen, 1995). The SOM was inspired by the way in which various human sensory impressions are neurologically mapped into the brain such that spatial or other relations among stimuli correspond to spatial relations among the neurons. In a SOM, the neurons (clusters) are organized into a grid--usually two-dimensional, but sometimes one-dimensional or three- or more-dimensional. The grid exists in a space that is separate from the input space; any number of inputs may be used. A SOM tries to find clusters such that any two clusters that are close to each other in the grid space have codebook vectors close to each other in the input space. Another way to look at this is that a SOM tries to embed the grid in the input space such that every training case is close to some codebook vector, but the grid is bent or stretched as little as possible. Yet another way to look at it is that a SOM is a (discretely) smooth mapping between regions in the input space and points in the grid space.



Let the codebook vectors be indexed by a subscript j , and let the index of the codebook vector nearest to the current training case be n . The Kohonen SOM algorithm requires a

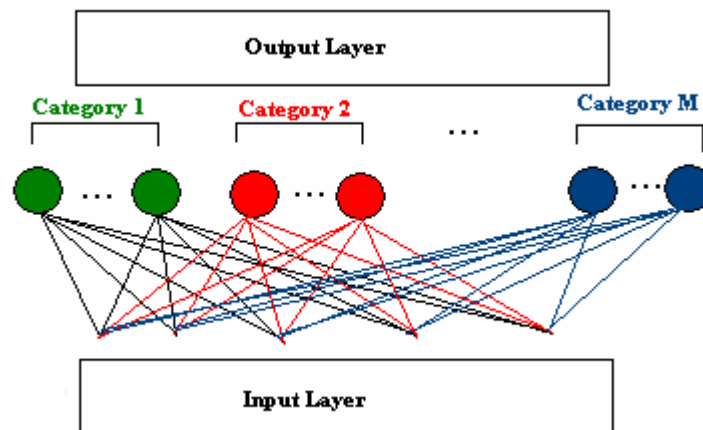
neighbourhood function $K(j, n)$, where $K(j, j) = 1$ and $K(j, n)$ is usually a non-increasing function of the distance (in any metric) between codebook vectors j and n in the grid space (not the input space). Usually, $K(j, n)$ is zero for codebook vectors that are far apart in the grid space. As each training case is processed, all the codebook vectors are updated as:

$$\text{new_codebook}_j = \text{old_codebook}_j * [1 - K(j, n) * \text{learning_rate}] + \text{data} * K(j, n) * \text{learning_rate}$$

The neighbourhood function does not necessarily remain constant during training. The neighbourhood of a given codebook vector is the set of codebook vectors for which $K(j, n) > 0$. To avoid poor results (akin to local minima), it is usually advisable to start with a large neighbourhood, and let the neighbourhood gradually shrink during training. If $K(j, n) = 0$ for j not equal to n , then the SOM update formula reduces to the formula for Kohonen vector quantization. In other words, if the neighbourhood size (for example, the radius of the support of the neighbourhood function $K(j, n)$) is zero, the SOM algorithm degenerates into simple VQ.

2.2.9 LVQ

Learning Vector Quantization--competitive networks for supervised classification (Kohonen, 1988, 1995; Ripley, 1996). Each codebook vector is assigned to one of the target classes. Each class may have one or more codebook vectors. A case is classified by finding the nearest codebook vector and assigning the case to the class corresponding to the codebook vector. Hence LVQ is a kind of nearest-neighbour rule.



Ordinary VQ methods, such as Kohonen's VQ or k-means, can easily be used for supervised classification. Simply count the number of training cases from each class assigned to each cluster, and divide by the total number of cases in the cluster to get the posterior probability.

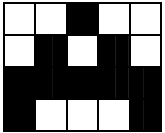
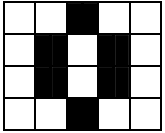
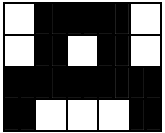
For a given case, output the class with the greatest posterior probability--i.e. the class that forms a majority in the nearest cluster. Such methods can provide universally consistent classifiers even when the codebook vectors are obtained by unsupervised algorithms. LVQ tries to improve on this approach by adapting the codebook vectors in a supervised way. There are several variants of LVQ--called LVQ1, OLVQ1, LVQ2, and LVQ3--based on heuristics.

3 Data Preprocessing & Feature Extraction

Appropriate data presentation and preprocessing is often essential for accurate classification results. For best models, inputs have to be presented as either Boolean values or real values of a fixed dimensionality.



We can construct a pattern vector from almost all kinds of data objects by selecting the relevant properties of these objects. Here, we give several examples how this can be achieved for concrete settings.

In character reorganization, a monochrome (1-bit per pixel) image of alphabets can be obtained via scanned or photo camera. The mapping of character image (an object) to the pattern vector can be done on a quantitative basis as following

Data Object	Pattern Matrix	Pattern with 20 attributes. a1, a2, a3... a20	Pattern with class label
 Monochrome image of alphabet "A".	00100 01010 11111 10001 5x4 matrix	00100010101111110001 20 dimension pattern without class label	00100010101111110001A 20 dimension pattern with class label
 Monochrome image of alphabet "O".	00100 01010 01010 00100 5x4 matrix	00100010100101000100 20 dimension pattern without class label	00100010100101000100O 20 dimension pattern with class label
 Monochrome image of alphabet "A".	01110 01010 11111 10001 5x4 matrix	01110010101111110001 20 dimension pattern without class label	01110010101111110001A 20 dimension pattern with class label

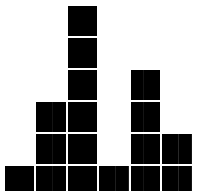
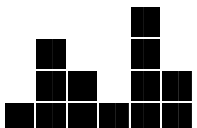
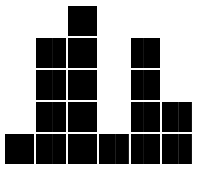
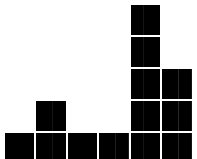
Here we have 5x4 monochrome images forming a matrix of binary-valued pixels. We have represented all blacked pixels as 1 and white as 0. Joining all matrix rows to one sequence would generate a pattern of 20 attributes i.e. dimension 20. We can also label this pattern to a class from where it belongs. The attributes have discrete values (0 or 1, Boolean entities) but if we would have considered colours of characters then attributes could have color values for each pixel. The important thing to note is that similar classes should have similar pattern vectors. In the above example, the patterns of class “A” are similar and they do not match the patten for class “O”.

It is not always possible to map quantitatively from a data object to a pattern matrix, we may also use some qualitative mappings.

Data Object	Pattern with 4 attributes. Living, Runs, Beautiful, Color	Pattern with class label
 Object of kind “Human”	Yes Feet Yes Brown	Yes Feet Yes Brown Human
 Object of kind “Car”	No Wheels No Brown	No Wheels No Brown Car

But these qualitative attributes can be understood well by a human, for pattern recognition methods such as neural networks, we must translate the qualitative to quantative attributes by giving some numerical values to each attribute. Sometimes attributes are missing and proper care should be taken to fill in any imaginary attribute.

Another example is the generation of Proteomic Patterns from human serum by mass spectroscopy. Proteomic pattern consist of a vector of intensity values for all reported wavelengths, typically several 100 to 1000 wavelengths heights are included. The bar graph values are obtained and mapping to pattern is done by taking only the values along the Y-axis because the values on the X-axis are constant in all cases.

Data Object	Pattern with 6 attributes. a1, a2, a3, a4, a5, a6	Pattern with class label
 <p>Spectrometric spectrum of "Cancer" patient</p>	<p>1 3 6 1 4 2</p> <p>6 dimensional pattern without class label</p>	<p>1 3 6 1 4 2 C</p> <p>6 dimensional pattern with class label</p>
 <p>Spectrometric spectrum of "Healthy" patient</p>	<p>1 3 2 1 4 2</p> <p>6 dimensional pattern without class label</p>	<p>1 3 2 1 4 2 H</p> <p>6 dimensional pattern with class label</p>
 <p>Spectrometric spectrum of "Cancer" patient</p>	<p>1 4 5 1 4 2</p> <p>6 dimensional pattern without class label</p>	<p>1 4 5 1 4 2 C</p> <p>6 dimensional pattern with class label</p>
 <p>Spectrometric spectrum of "Healthy" patient</p>	<p>1 2 1 1 5 3</p> <p>6 dimensional pattern without class label</p>	<p>1 2 1 1 5 3 H</p> <p>6 dimensional pattern with class label</p>

Given above were very simple toy examples about the construction of pattern vectors from data objects but in real life applications, it is not so simple; high dimensionality, noise, missing values etc might cause problems. Therefore, data preprocessing should be considered before implementing any kind of pattern recognition system, for example:

1. **Data conversion:** turning attribute data types into allowed representation;
2. **Data cleaning:** Filling in missing values, smoothing noisy data, identifying or removing outliers, and resolving inconsistencies;
3. **Data transformation:** mapping data to another representation, for example to a normalized one;
4. **Data discretization:** replacing numerical attributes with atomic states.
5. **Data dimensionality reduction:** discarding a subset of attributes.

3.1 Simplest Approach: Normalization, Standardization and Rescaling

The simplest form of data processing "**Rescaling**" a pattern vector means to add or subtract a constant and then to multiply or divide by a constant, as you would do to change the units of measurement of the data, for example, to convert a temperature from Celsius to Fahrenheit.

"**Normalizing**" a pattern vector most often means dividing by a norm of the vector, for example, to make the Euclidean length of the vector equal to one. In the NN literature, "normalizing" also often refers to rescaling by the minimum and range of the vector, to make all the elements lie between 0 and 1.

"**Standardizing**" a pattern vector most often means subtracting a measure of location and dividing by a measure of scale. For example, if the pattern vector contains random values with a Gaussian distribution, you might subtract the mean and divide by the standard deviation, thereby obtaining a "standard normal" random variable with mean 0 and standard deviation 1.

However, all of the above terms are used more or less interchangeably depending on the customs within various fields.

3.2 Complex Approach: Dimensionality reduction and feature extraction

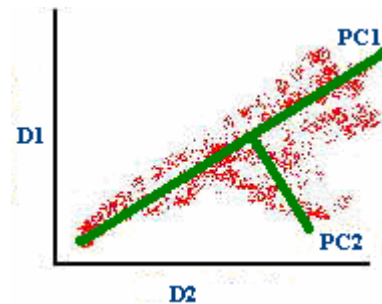
One of the most important forms of pre-processing involves a reduction in the dimensionality of pattern vectors. At the simplest level this could involve discarding a subset of the original vector. The principle motivation for dimensionality reduction is that it can help to alleviate the worst effects of the curse of dimensionality. Dimensionality reduction should be approached with caution because it discards information. If that information is irrelevant, then dimensionality reduction can be quite helpful but if that information is important, it can be disastrous for pattern recognition system's reliability.

The **curse of dimensionality** refers to the exponential growth of a hypervolume as a function of dimensionality. The curse of dimensionality causes pattern recognition system with lots of irrelevant inputs to behave relatively badly: the dimension of the input space is high, and the pattern recognition system uses almost all its resources to represent irrelevant portions of the space. A partial remedy is to preprocess the input in the right way, for example by scaling the components according to their "importance". Even if we have a pattern recognition algorithm which is able to focus on important portions of the input space, the higher the dimensionality of the input space, the more data may be needed to find out what is important and what is not. So to reduce the dimension of a pattern vector and the selection of important features becomes important. It can be done by PCA, for example.

3.2.1 Dimensionality reduction with Principle Components Analysis (PCA)

PCA is commonly used in micro array research as a cluster analysis tool. It is designed to capture the variance in a dataset in terms of **principle components**. In effect, one is trying to reduce the dimensionality of the data to summarize the most important (i.e. defining) parts whilst simultaneously filtering out noise. Normalization, however, can sometimes remove this noise and make the data less variate, which could affect the ability of PCA to capture data structure.

Principle Components are a set of variables that define a projection that encapsulates the maximum amount of variation in a dataset and is orthogonal (and therefore uncorrelated) to the previous principle component of the same dataset.



The lines represent 2 consecutive principle components. Note that they are orthogonal (at right angles) to each other. PCA can be imposed on datasets to capture the cluster structure prior to clustering. Let us consider an example:

Suppose a microarray dataset composed of 1000 genes, each of which have an expression value over 10 experiments. The dimensionality of that dataset is therefore 10 (i.e. there are 10 axes). The data though clumped around several central points in that hyperspace will

generally tend towards one direction. If one were to draw a solid line that best describes that direction, then that line is the first principle component (PC). Any variation that is not captured by that first PC is captured by subsequent orthogonal PCs. The first 3 PCs could themselves act as Cartesian axes. The data they capture can therefore be plotted in terms of these axes. Hence there is a reduction of dimensionality. When the data is plotted in this manner they are said to be plotted in PC-space.

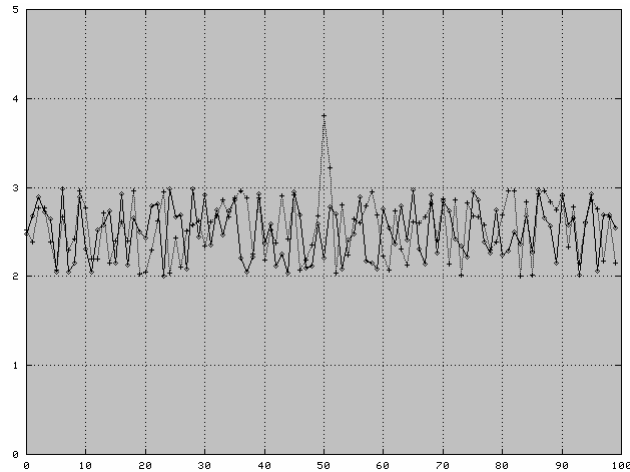
3.2.2 Feature Extraction

In some sense many neural networks perform a kind of automatic feature extraction. We do not choose the features the network will train to and these feature may or may not have desirable properties. They are features in a fairly abstract sense and are unlikely to resemble anything a human would recognize as a feature such as line segments or loops or arcs. Therefore we would like to examine the use of explicit feature selection and feature extraction in our desire to achieve a reduction in dimensionality.

For example, tall people tend to be stronger than short people. There are several reasons for this: tall people have longer arms and legs, which gives their muscles more mechanical advantage; tall people tend to have bigger muscles, simply because they are bigger people; and tall people tend to be men, who have higher testosterone levels, which helps them build more muscle. The fact remains, however, that some short women can lift more weight than some tall men. So if we were to try to classify people into two groups, 'strong' and 'weak', without actually measuring how much they can lift, height might be one feature we would use as a predictor. But, it couldn't be the only one if we wanted our classification to be highly reliable.

If a single feature is not a good class predictor on its own, the alternative is to look for one or more **sets of features** that together make a good predictor of what class an object falls into. For example, neither height nor weight are particularly good predictors of obesity; but taken together, they predict it fairly well.

It is worth making concrete what is meant by a feature extraction from gene expression data that is application to bioinformatics. The figure below shows two genes with 100 samples each. One gene (let's call it Gene A) clearly has an enhanced expression value around sample 50. This expression level 'bump' can be considered as a feature. If every gene expression profile from tissues of the same class showed the same bump, this feature would be a good predictor of what tissue class a new sample of tissue belonged to.



Suppose we observed the following data:

Tissue Class	Average Expression Level	
	Gene A, Sample 50	Gene B, Sample 50
Normal	3.5	2.5
Cancer	2.5	2.5

In this case, Gene A has a feature, an enhanced expression level for sample 50, that is a good predictor of which class (Normal or Cancer) a tissue belongs to. Gene B has no such feature; its average expression level at sample 50 is independent of tissue class.

3.3 Generalization

The learning result of a classifier highly depends upon the training set. With a good training set and an effective algorithm, a classifier classifies inputs not belonging to the training set very effectively. The training set has to be large and diverse to adequately represent the problem domain. The term training set is used to refer to a set of input pattern vectors for use in training a classifier to learn over a data model. Usually, a large number of training patterns would be used in the training of any genuinely useful classifier. The goal of network training is not to learn an exact representation of the training data itself, but rather to build a model of the process which generates data. The critical issue in developing a neural network is generalization: how well will the network make predictions for cases that are not in the training set.

Generalization means that when a classifier learns over a training set (examples), what the classifier is really learning is not the specific training set but the general principles that control or determine the answers to that training set. In other words, a classifier does not merely learn cases but extracts the relevant features that distinguish the examples in a training set and

absorbs those features. Moreover, a classifier does so without being told what the relevant features are. The training process permits the network to determine for it what features are the key determining ones.

Generalization demands a significant number of different examples. If number of examples is too small then what a classifier can do is to memorize them. This means the classifier's opinion on new examples may or may not have much relevance. Thus, one has to take care for a good generalization ability of the classifier. Other keywords closely related to the generalization ability of a classifier are the terms over and under fitting.

NNs can suffer from either **underfitting** or **overfitting**. A network that is not sufficiently complex can fail to detect fully the signal in a complicated data set, leading to underfitting. A network that is too complex may fit the noise, not just the signal, leading to overfitting. Overfitting occurs when the network has so much information processing capability that it will learn insignificant aspects of the training sets. Overfitting is especially dangerous because it can easily lead to predictions that are far beyond the range of the training data with many of the common types of NNs. Overfitting can also produce wild predictions in multilayer perceptrons even with noise-free data. The best way to avoid overfitting and underfitting is to use lots of training data or other regularization schemes.

Some conditions for good generalization to avoid overfitting include:

- The inputs to the network contain sufficient information pertaining to the target, so that there exist a mathematical function relating correct outputs to inputs with the desired degree of accuracy. One cannot expect a network to learn a nonexistent function. Finding good inputs for a network and collecting enough training data often take far more time and effort than training the network.
- The learning function (that relates inputs to correct outputs) should be smooth. In other words, a small change in the inputs should, most of the time, produce a small change in the outputs.
- The training cases constitute a sufficiently large and representative subset of the set of all cases that we want to generalize.

Split-sample or hold-out validation, **Cross-validation** and **Bootstrapping** are the methods for estimating the generalization error based on "resampling". The resulting estimates of the generalization error are often used for choosing among various models, such as different network architectures. The methods are thereby statistically reliable and do not depend on specifics of the training set.

3.3.1 Split-sample or hold-out validation

The most commonly used method for estimating the generalization error in neural networks is to reserve a part of the data as a "test" set, which must not be used in any way during training. The test set must be a representative sample of the cases that you want to generalize to. After training, run the network on the test set, and the error on the test set provides an unbiased estimate of the generalization error, provided that the test set was chosen randomly. The disadvantage of the split-sample validation is that it reduces the amount of data available for both training and validation.

The holdout method is the simplest kind of **cross validation**. The data set is separated into two sets, called the training set and the testing set. The function approximator fits a function using the training set only. Then the function approximator is asked to predict the output values for the data in the testing set (it has never seen these output values before). The errors it makes are accumulated as before to give the mean absolute test set error, which is used to evaluate the model.

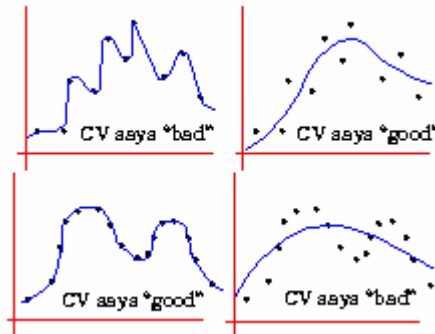
3.3.2 Cross-validation

Cross-validation is the practice of partitioning a sample data into subsamples such that the analysis is initially performed on a single subsample, while further subsamples are retained "blind" for subsequent use in confirming and validating the initial analysis. The distinction between cross-validation and split-sample validation is extremely important because cross-validation is markedly superior for small data sets. It achieves better statistical reliability for these situations.

K-fold cross validation is one way to improve over the holdout method. The data set is divided into k subsets (approximately equal size) and the holdout method is repeated k times. Each time, one of the k subsets is used as the test set and the other $k-1$ subsets are put together to form a training set. Then the average error across all k trials is computed. The advantage of this method is that it matters less how the data gets divided. Every data point gets to be in a test set exactly once, and gets to be in a training set $k-1$ times. The variance of the resulting estimate is reduced as k is increased. The disadvantage of this method is that the training algorithm has to be rerun from scratch k times, which means it takes k times as much computation to make an evaluation.

Leave-one-out cross validation is K-fold cross validation taken to its logical extreme, with K equal to N , the number of data points in the set. That means that N separate times, the function approximator is trained on all the data except for one point and a prediction is made for that point. As before the average error is computed and used to evaluate the model. The evaluation given by leave-one-out cross validation error is good, but at first pass it seems very expensive to compute. Fortunately, locally weighted learners can make Leave-out-one predictions just as easily as they make regular predictions. If k equals the sample size, this is

called "leave-one-out" cross-validation. "Leave-v-out" is a more elaborate and expensive version of cross-validation that involves leaving out all possible subsets of v cases. The figure below shows an example how a model can be reached by such methods in comparison to the training error. The data set in the top two graphs is a simple underlying function with significant noise. Cross validation tells us that broad smoothing is best. The data set in the bottom two graphs is a complex underlying function with no noise. Cross validation tells us that very little smoothing is best for this data set.



3.3.3 Bootstrapping

Bootstrapping seems to work better than cross-validation in many cases. In the simplest form of bootstrapping, instead of repeatedly analyzing subsets of the data, repeated subsamples of the data are chosen and used for training. Each subsample is a random sample with replacement from the full sample. Depending on anywhere from 50 to 2000 subsamples might be used. There are many more sophisticated bootstrap methods that can be used not only for estimating the generalization error but also for estimating confidence bounds for the network outputs. For estimating the generalization error in classification problems, the .632+ bootstrap (an improvement on the popular .632 bootstrap) is one of the currently favoured methods that have the advantage of performing well even when there is severe overfitting.

4 Performance Measures

To find the performance quality of machine learning algorithms is not always simple. In a simplest way, we can just calculate the percentage of correct predictions an algorithm has made and compare it with another algorithm. But if we are really interested in the overall performance of an algorithm, then we must measure the performance against a tolerance point. There exist a number of standard methods to measure the performance of classification techniques.

4.1 Percent Correct Method

This is a simplest way to determine the classifier's performance. If **N** is the total number of examples and **K** is the number of random examples selected for the training such that they are distributed over each class being represented where $K < N$ then the number of examples belonging to the test set is equal to $N - K$. After training of the network, examples from test set ($N - K$) are given to network for classification. Lets say that the network classify correctly **C** number of examples from the test set then **Percent Correct** = $(C \times 100) / (N - K)$. It is not fare to test an algorithm with the same set of pattern used to train it.

4.2 Receiver Operating Characteristic (ROC)

Binary classification is the task of classifying the members of a given set of objects into two groups on the basis of whether they have some property or not. Some typical binary classification tasks are 1) medical testing to determine if a patient has certain disease or not (the classification property is the disease) 2) quality control in factories; i.e. deciding if a new product is good enough to be sold, or if it should be discarded (the classification property is being good enough) 3) deciding whether a page or an article should be in the result set of a search not (the classification property is the relevance of the article - typically the presence of a certain word in it). A first performance measure for binary classifier is offered by the ROC analysis.

Receiver Operating Characteristic ROC curves are used to evaluate the results of a prediction. ROC is typically evaluated by a **confusion matrix** as illustrated in the figure below for only 2 class problems. The columns are the Predicted class and the rows are the Actual class. In the confusion matrix, **TN** is the number of negative examples correctly classified (True Negatives), **FP** is the number of negative examples incorrectly classified as positive (False Positives), **FN** is the number of positive examples incorrectly classified as negative (False Negatives) and **TP** is the number of positive examples correctly classified

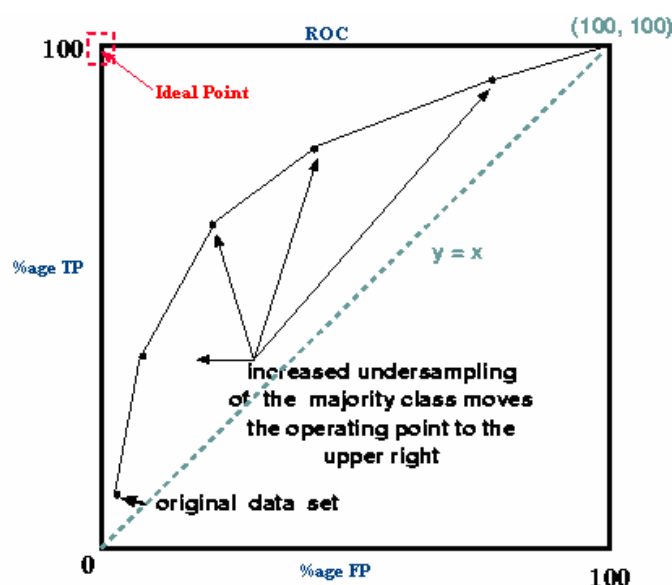
(True Positives). For example, let us say, we test some people for the presence of a disease. Some of these people have the disease, and our test says they are positive. They are called true positives TP. Some have the disease, but the test claims they don't. They are called false negatives FN. Some don't have the disease, and the test says they don't - true negatives TN. Finally, we might have healthy people who have a positive test result false positives FP.

The predictive accuracy is the performance measure generally associated with machine learning algorithms and is defined as **Accuracy** = $(TP+TN)/(TP+FP+TN+FN)$. In the context of balanced datasets and equal error costs, it is reasonable to use error rate as a performance metric whereby **Error rate**=1-Accuracy. In the presence of imbalanced datasets with unequal error costs, it is more appropriate to use the ROC curve.

	Predicted negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Confusion Matrix

ROC curves can be thought of as representing the family of best decision boundaries for relative costs of TP and FP. On the ROC curve the X-axis represents $\%FP = FP/(TN+FP)$ and the Y-axis represents $\%TP = TP/(TP+FN)$. The ideal point on the ROC curve would be (0,100) that are all positive examples are classified correctly and no negative examples are misclassified as positive. One way the ROC curve can be swept out is by manipulating the balance of training samples for each class in the training set. The figure below shows an illustration. The line $y = x$ represents the scenario of randomly guessing the class.



The area under the ROC Curve (AUC) is a useful metric for the classifier performance as it is independent of the decision criterion selected and prior probabilities. The AUC comparison can establish a dominance relationship between classifiers. If the ROC curves are intersecting, the total AUC is an average comparison between the models. However, for some specific cost and class distributions, the classifier having maximum AUC may in fact be suboptimal. Hence, we also compute the ROC convex hulls, since the points lying on the ROC convex hull are potentially optimal.

4.3 Specificity and Sensitivity

Sensitivity is the proportion of people that are tested positive of all the positive people tested; that is (true positives) / (true positives + false negatives) i.e. **Sensitivity=TP/(TP+FN)**. It can be seen as the probability that the test is positive given that the patient is sick. The higher the sensitivity, the less real cases of diseases goes undetected (or, in the case of the factory quality control, the less faulty products go to the market). The sensitivity of a binary classification test or algorithm, such as a blood test to determine if a person has a certain disease, or an automated system to detect faulty products in a factory, is a parameter that expresses something about the test's performance. The sensitivity of such a test is the proportion of those cases having a positive test result of all positive cases (e.g., people with the disease, faulty products) tested. A sensitivity of 100% means that all sick people or faulty products were recognized as such, but it alone doesn't tell us all about the test, as a 100% sensitivity can be trivially achieved by labelling all test cases positive, despite their true status.

Specificity is the proportion of people that are tested negative of all the negative people tested; that is (true negatives) / (true negatives + false positives) i.e.

Specificity=TN/(TN+FP). As with sensitivity, it can be looked at as the probability that the test is negative given that the patient is not sick. The higher the specificity, the less healthy people are labelled as sick (or, in the factory case, the less money the factory loses by discarding good products instead of selling them).

In theory, sensitivity and specificity are independent in the sense that it is possible to achieve 100 % in both. In practice, there often is a tradeoff, and you can't achieve both. In addition to sensitivity and specificity, the performance of a binary classification test can be measured with **positive and negative prediction values**. These are possibly more intuitively clear: the positive prediction value answers the question "how likely it is that I really have the disease, given that my test result was positive? ". **Positive Prediction Values PPV= TP/(TP+FP)** and **Negative Prediction Values NPV= 1-PPV**.

Sensitivity and Specificity are independent from the population in the sense that they don't change depending on what the proportion of positives and negatives tested are. Indeed, one

can determine the sensitivity of the test by testing only positive cases. However, the prediction values are dependent on the population. As an example, say that we have a test for a disease with 99 % sensitivity and 99 % specificity. Say we test 2000 people, and 1000 of them are sick and 1000 of them are healthy. You are likely to get about 990 true positives, 990 true negatives, and 10 of false positives and negative each. The positive and negative prediction values would be 99 %, so the people can be quite confident about the result. Say, however, that of the 2000 people only 100 are really sick. Now you are likely to get 99 true positives, 1 false negative, 1881 true negatives and 190 false positives. Of the 190+99 people tested positive, only 99 really have the disease - that means, intuitively, that given that your test result is positive, there's only 34.2 % chance that you really have the disease. On the other hand, given that your test result is negative, you can really be reassured: there's only 1 chance in 1881, or 0.05% probability, that we have the disease despite of our test results.

5 Pattern Recognition with Dynamic-GRLVQ Algorithm

Kohonen proposed Learning Vector Quantization LVQ [1], a simple and perceptive though very robust prototype based clustering algorithm which we already introduced in chapter 2. Originally, LVQ is a very intuitive but purely heuristic training algorithm which is based on prototype based classification. For one of the next versions of LVQ, Generalized Learning Vector Quantization GLVQ [2] avoids numerical instability due to stochastic gradient decent on a cost function. Combining adaptive relevance factors with GLVQ to form GRLVQ [3] [4] has been a big achievement to improve the classification accuracy by faster convergence and by better adaptation. An extension to GRLVQ is Supervised Relevance Neural Gas SRNG [5] that assures a distribution of prototypes among the data set at the beginning in such a way that a parameterized cost function is minimized through learning rule.

GRLVQ is very stable with more adaptive parameters (an adaptive metric) and showed in several tasks competitive results to SVM [20]. GRLVQ is also a large margin classifier, thus it generalizes very nicely to new data points. The resulting classifier is more natural than SVM, since it represents the classifier in terms of typical datapoints instead of datapoints lying at the border. The size of the classifier is usually much smaller than the classifier size obtained by SVM. GRLVQ training is a stochastic gradient descent, i.e. convergence to global optima is not guaranteed. But usually it works reasonably fast. GRLVQ has very good generalization, flexible due to adaptive metric, small classifier, intuitive but only local optimal training possibly and perform only classification.

Some of the usual problems for LVQ based methods are that one cannot optimally guess about the number of prototypes required for initialization for multimodal data structures i.e. these algorithms are very sensitive to initialization of prototypes and one has to pre define the optimal number of prototypes before running the algorithm. If a prototype, for some reasons, is ‘outside’ the cluster which it should represent and if there are points of a different categories in between, then the other points act as a barrier and the prototype will not find its optimum position during training.

Since the model complexity is not known in many cases, we avoid this problem by introducing a dynamic version of GRLVQ, Dynamic-GRLVQ (DGRLVQ), which adapts the model complexity to the given problem during training i.e. during training, it adds or removes prototypes dynamically one by one for each category until satisfactory classification results are achieved. New prototypes are initialized on either the ‘mean position’ of misclassified points or can be initialized on the ‘first misclassified point’. In addition to this, dynamically

removal of stray prototypes is introduced to remove all prototypes those are outside from the cluster which they must belong. One reason could be that the prototype is allocated to a noise point in another cluster and this prototype is continually being outside from its cluster and never wins, the survival rate of such kind of prototypes start decreasing and over some threshold these prototypes are qualified to get removed.

The prototype removal feature is important to remove a prototype allocated to points subject to noise in other clusters. In other words, a prototype allocated to points squeezed between clusters of a wrong category can be removed. Care has been taken for not removing prototypes more than some certain percentage of the total prototype population of same category.

Prototypes can also dynamically adapt the distance matrices. In addition to this, dynamically changing the exponents of the relevance vector and exponents of coordinates is also possible. Learning has been made local to prototypes in Dynamic-GRLVQ i.e. every newly added prototype has its own local learning rates (local learning rate correct and local learning rate wrong) and the learning rate of relevance factor is made global that can also be dynamically changed on a run. In other words, there are mainly three kinds of learning rates and all learning rates can be decreased with respect to some percentage after each update.

1. Learning rate correct; a local learning rate of a prototype by which it learns (comes closer to) a cluster it represents.
2. Learning rate wrong; a local learning rate of a prototype by which it learns (moves away from) a cluster it does not represent.
3. Learning rate lambda; a global learning rate of relevance factors.

It is very important for a prototype to obey local learning rates (correct & wrong) and not a global one because whenever a new prototype is added, we expect it runs toward a cluster where it belongs and away from a cluster where it does not belong. This is only possible when all prototypes have their own local learning (correct & wrong) rates those may decrease over time. Since already trained prototype (old ones) are likely located within there cluster whereas new ones are not, the learning rate for relevance factors stays on a global level to ensure the equal contribution of each prototype to the global weighting scheme.

In the beginning, only one prototype is assigned per class. Prototypes are initiated in the class centre. As the update cycles processes, we keep track of error rate. If the error rate keeps on decreasing then update cycles may keep on proceeding but if the error rate becomes stable or starts increasing, we add a new prototype for misclassified points per class. Moreover we can also add a prototype after some fixed number of update cycles. DGRLVQ represents how an optimal model can be obtained from dynamic increasing and shrinking network sizes. It is about maintaining prototype communities for clusters.

5.1 Basic DGRLVQ

We now introduce the model in an exact way.

A set of an output category/class is defined as

$$L = \{0, 1, \dots, i, \dots, (e-1)\}$$

where number of categories is $\#L = e$ and any element $i \in \mathbb{N}$

An input point, pattern vector, is given by

$$v = (x_0, x_1, \dots, x_j, \dots, x_{n-1})$$

where the dimension of v is $D_v = n$ and any element $x_j \in \mathbb{R}$

A pattern vector matrix of a category $i \in \mathbb{N}$ is given by

$$V_i = \{v_0, \dots, v_{k-1}\}$$

where k is the total number of pattern vectors and any row vector element of V is represented simply as v

Let $c_v \in L$ be the label of input v

Let $W = \{w_r\}$ be the set of all codebook vectors $w_r \in \mathbb{R}^n$ and $c_r \in L$ be the category label of w_r and $W_c = \{w_r \mid c_r = c\}$ be the subset of vectors assigned to class $c \in L$

Any prototype can be represented as $s = \{w, c, d^\lambda, q, \varepsilon^+, \varepsilon^-\}$ where w is the weight vector, c is the category of prototype, d^λ is the global distance metric, ε^+ is the learning rate correct, ε^- is the learning rate wrong, q is the qualifying winner count. It is important to maintain q above some certain threshold t or else this prototype may be removed i.e. to remove a prototype $q < t$ applies.

Let A be the set of all prototypes which is dynamic. For a given $v \in \mathbb{R}^n$, classification in DGRLVQ takes place according to the winner take all rule. That is for a mapping function Ω , vector $v \in V$ is mapped onto that prototype $s \in A$ with distance

$$\Omega_{V \rightarrow A}^\lambda : v \mapsto s(v) = \arg \min_{r \in A} d_{r \in A}^\lambda(v, w_r) \quad (1)$$

to be minimum and prototype s is called winner. Here $d^\lambda(v, w)$ being an arbitrary differential distance measure depending upon relevance vector λ . The subset of the input space

$$\Pi_r^\lambda = \{v \in V : r = \Omega_{V \rightarrow A}^\lambda(v)\} \quad (2)$$

which is mapped to a particular prototype according to (1), forms the masked Π_r^λ receptive field of that neuron, which is the set of points for which the prototype becomes winner. The training algorithm adapts the prototypes such that for each category $c \in L$ the corresponding codebook vector W_c represents the class as accurately as possible. This means that the set of points in any given class $V_c = \{v \in V \mid c_v = c\}$ and the union $\bigcup_c = \bigcup_{r \mid w_r \in w_c} \Pi_r^\lambda$ of the receptive fields of the corresponding prototypes should differ as little as possible.

Let $f(x) = (1 + \exp(-x))^{-1}$ be the logistic function then DGRLVQ minimizes the cost function

$$C_{DGLVQ} = \sum_v f(\mu_\lambda(v)) \quad (3)$$

$$\text{with } \mu_\lambda(v) = \frac{d_{r+}^\lambda - d_{r-}^\lambda}{d_{r+}^\lambda + d_{r-}^\lambda}$$

where d_{r-}^λ is the distance of the closest prototype from a vector and d_{r+}^λ is the distance of closest prototype from a vector of another category.

Using $\frac{\partial \mu_\lambda(v)}{\partial w_{r+}} = \xi^+ \frac{\partial d_{r+}^\lambda}{\partial w_{r+}}$ and $\frac{\partial \mu_\lambda(v)}{\partial w_{r-}} = \xi^- \frac{\partial d_{r-}^\lambda}{\partial w_{r-}}$ with

$$\xi^+ = \frac{2 \cdot d_{r-}^\lambda}{(d_{r+}^\lambda + d_{r-}^\lambda)^2} \quad (4)$$

and

$$\xi^- = \frac{2 \cdot d_{r+}^\lambda}{(d_{r+}^\lambda + d_{r-}^\lambda)^2} \quad (5)$$

one obtains for the weights update

$$\Delta w_{r+} = \varepsilon^+ \cdot f' | \mu_{\lambda}(v) \cdot \xi^+ \cdot \frac{\partial d_{r+}^{\lambda}}{\partial w_{r+}} \quad (6)$$

where $\varepsilon^+ > 0$ is the learning rate correct which is local for each prototype

ε^+ decays as $\varepsilon_{new}^+ = \varepsilon_{old}^+ - \alpha \cdot \varepsilon_{old}^+ / 100$, s being winner $q_{new} = q_{old} + 1$

$$\Delta w_{r-} = -\varepsilon^- \cdot f' | \mu_{\lambda}(v) \cdot \xi^- \cdot \frac{\partial d_{r-}^{\lambda}}{\partial w_{r-}} \quad (7)$$

where $\varepsilon^- > 0$ is the learning rate wrong which is local for each prototype s

ε^- decays as $\varepsilon_{new}^- = \varepsilon_{old}^- - \alpha \cdot \varepsilon_{old}^- / 100$, s being loser $q_{new} = q_{old} - 1$

For the adaptation of the parameter λ_k we get

$$\Delta \lambda_k = -\varepsilon \cdot f' | \mu_{\lambda}(v) \cdot \frac{\partial \mu_{\lambda}(v)}{\partial \lambda_k} \quad (8)$$

with

$$\frac{\partial \mu_{\lambda}(v)}{\partial \lambda_k} = \xi^+ \frac{\partial d_{r+}^{\lambda}}{\partial \lambda_k} - \xi^- \frac{\partial d_{r-}^{\lambda}}{\partial \lambda_k} \quad (9)$$

where $\varepsilon > 0$ is the global learning rate for the relevance vector

ε decays as $\varepsilon_{new} = \varepsilon_{old} - \alpha \cdot \varepsilon_{old} / 100$

The relevance term is normalized enforcing $\lambda_0 + \lambda_1 + \dots + \lambda_{n-1} = 1$ and $\lambda_i \geq 0$

The qualifying winner count q increases when a prototype s is a winner and decreases in the other case. Each prototype s must maintain the qualifying winner count q above the threshold t for its survival or else this prototype may be removed i.e. to remove a prototype $q < t$ applies. Before adding a new prototype s into the dynamic set A of all prototypes, we remove all those prototypes from A with $q < t$ or we remove all stray prototypes which are representing no vector. Now, we set $q = t$ and then add a prototype s into the dynamic set A

As we intend to remove the prototypes with $q < t$, there could be situations when all (or almost all) the prototypes of same category exhibits $q < t$ and lead to removal of all of them.

To avoid such kind of situation, we introduce another threshold factor b_c , which is the total number of prototypes of category c allowed to be removed for each update.

For a category c , if m_c is a maximum number of prototypes in A , j_c is the percentage of prototypes that may be removed from A , the total number of prototypes allowed to be removed is given as

$$b_c = \frac{m_c \cdot j_c}{100} \quad (10)$$

After removal of the prototypes, the dynamic set A is updated as

$$A_{new}^m = A_{old}^{m_c} \setminus A_{old}^{b_c} \quad (11)$$

And after adding prototypes, the dynamic set A is updated as

$$A_{new}^m = A_{old}^{m_c} \cup s_c^q \quad (12)$$

For a category c , $O_c = \{v_1, v_2, \dots, v_r\}$ is a matrix of r misclassified vectors, then the addition of new prototype s_c can be carried out at a mean position given by

$$p_c = \frac{v_1 + v_2 + \dots + v_r}{r} \quad (13)$$

Another way of addition of a new prototype s_c is a first come first served method that can be carried out by assigning s_c at a position of the first misclassified vector v_1 in the set

$V_c = \{v_1, v_2, \dots, v_r\}$ or by selecting any random misclassified vector v_i , but this way of adding new prototype is ambiguous because we may be assigning a prototype to an outlier.

However, the first come first served approach is faster as we need not to find a mean position.

With the initialization of the algorithm we keep track of the classification error CE. CE is equal to the sum of all misclassified vectors of all categories divided by the total number of vectors of all categories.

$$CE = \frac{\#O_0 + \#O_1 + \dots + \#O_{e-1}}{\#V_0 + \#V_1 + \dots + \#V_{e-1}} \quad (14)$$

CE is expected to decrease with each update and it is very important factor to decide when to

add a new prototype for the misclassified classes. At algorithm initialization, CE is noted and as adaptation takes place, CE starts decreasing. If we find that CE is constant or increasing, we add new prototypes for each misclassified class.

For each update, all the training set is feed to the algorithm with a random selection of examples. The training set consists of an equal number of examples from all classes. A copy of the examples is made if a certain class has fewer examples compared to other. In other words, training set has an equal number of examples from all classes and these examples are fed to the algorithm randomly.

The algorithm stops if, the update cycles exhaust a predefined value or, CE becomes almost zero or, we terminated the algorithm prematurely.

Although we can measure the algorithm performance with standard methods, we have also embedded a probabilistic classification model in the algorithm based upon sigmoid density function. In our model, we have a set of output classes $L = \{0, 1, \dots, i, \dots, (e-1)\}$ where number of categories is $\#L = e$ and any element $i \in \mathbb{Z}$

If $v \in \mathbb{R}^n$ is a point of unknown category. Let p_c be the closest prototype of category $c \in L$ and p_i is the second closest prototype of category $i \in L$. If $d^\lambda(v, p_c)$ is the distance between v and p_c , $d^\lambda(v, p_i)$ is the distance between v and p_i than $d^\lambda(v, p_c) < d^\lambda(v, p_i)$ holds. We define the probability that v belongs to category $c \in L$ as

$$P(v \in c) = \frac{1}{1 + e^{-\sigma}} \quad (15)$$

where

$$\sigma = \frac{d^\lambda(v, p_c) - d^\lambda(v, p_i)}{d^\lambda(v, p_c) + d^\lambda(v, p_i)} \quad (16)$$

is chosen such that

$$P(v \in c) = 0.5 \quad (17)$$

Equation (15) specifies sigmoid probability density function centred at p_c where the midpoint between p_c and p_i has a 50% probability of being classified into either class c or i .

The classification threshold $t \in [0.5, 1.0]$ is chosen such that the classification is made if and only if

$$P(v \in c) < t \quad (18)$$

If equation 18 is not satisfied then we consider v being ambiguous and classification is not made. A tradeoff is thus offered between number of points classified and the accuracy of classification. Values of t approaching to 1.0 means more points to be classified at the cost of lower confidence in classification. Similarly, small values of t approaching to 0.5 classify fewer samples with high confidence.

5.2 Pseudocode

On an abstract level, DGRLVQ is formulated as the following algorithm:

1. Start with initialization of the total number of updates to be made, assign the learning rates and initialize one prototype per class at random position (Initializing prototypes for first time at random positions (and not at cluster centre) has an advantage that we need not to find the biggest cluster and its centre because prototypes would settle themselves automatically at generic position on first initialization after some steps)
2. Train the classifier with an update rules for a number of steps with update rules for distance matrices (stochastic, gradient decent etc) and learning adaptations.
3. If the classification is not yet ok, the error (CE) is still high, a predefined accuracy has not reached, the improvement of the last added round of prototypes has been large etc
 - a. Remove all those prototypes with less than threshold qualifying winner count i.e. remove all those prototypes, the number of points for which they are winners is smaller than some predefined threshold. Remove all stray prototypes those represents no cluster points.
 - b. Create a new prototype for each class for which points are misclassified, initialize this prototype in the centre of still misclassified points OR initialize this prototype at the position of a still misclassified point
 - c. goto 2

In Pseudocode, this reads as follows:

START

```
Initiate tm= total maximal modules;
// modules, this variable can be dynamically changed

Initiate tc= total maximum number of cycles;
//total cycles, this variable can be dynamically changed
//making tm=tc=0 meets the stop condition that can be done dynamically on a run

Loop till (tc > 0)
{
  Loop till (tm>0)
  {
    tm= tm-1;
    set C= Category selected with equal selection probability from category set L;
    set P= Pattern vector selected of kind C with equal probability from set V;
    train all prototypes for selected C & P;
    update prototype local learning rates(correct & wrong) with %age decay factor;
    update Learning rate of adaptive relevance vector with %age decay factor;
    update qualifying winner count for each prototype;
    set CE= classification error;
    if(CE=0)
    {
      tm=0;
      tc=0; //Stoping condition;
    }
    Else if ( tm<= 0 )
    {
      remove all loser prototypes with qualifying winner count less then threshold and all stray prototypes;
      add new Prototype Dynamically for all misclassified patterns( at centre or at first misclassified pattern);
    }
  }

  tc= tc-1;
  tm= total maximal modules;
}
```

END

6 Applications and Experiments

The DGRLVQ algorithm will be applied for classification tasks in different application areas in bioinformatics, especially cancer prediction, and other toy problems for demonstration purpose. In this section, we report on the performance of DGRLVQ and compare the results, where possible, to other algorithms.

Bioinformatics data may exhibit inhomogeneous class distributions. Some classes may have only a few numbers of data points whereas other class data points may be dominantly available. For DGRLVQ, such a class distribution imbalance would yield a biased prototype update because the frequent presentation of data points those belong to the largest class would implicitly push away prototypes of smaller classes in the neighbourhood. Therefore, we have used training sets having equal number of examples for each class. Equalization is done by making copies of data points of classes with few examples.

DGRLVQ algorithm is verified to confirm that it will properly classify samples that were not used in training. To test a classification algorithm, it is essential to perform multiple leave-out experiments, each with a different split between the training and testing (masked) sets. Often, there exists a split of samples into training and testing sets that performs significantly better than others [21]. The performance statistics of the classifier against multiple different splits is therefore reported. If the number of samples is significantly less than we can also reduce the number of multiple leave-out experiments and vice versa.

We have developed java DGRLVQ simulator known as Cool-D v1.0 and all the experimental results are based on this software. The software package can be obtained by contacting the author and is distributed under the GNU Public License (GNU).

6.1 Artificial Geographical satellite data

This is a two class geographical satellite dataset. We have generated points on the X-Y plane representing two classes of regions (forest and sea). Both of these classes overlap at some places and they cannot be linearly separated by a single line. This is a simple problem that is chosen to give a working overview of the DGRLVQ algorithm.

The class “forest” has 1275 xy coordinate points and the class “water” has 2040 xy coordinate points. We randomly picked 25% of training data points(319 data vectors) of the class “forest” and 25% of the training data point(510 data vectors) of the class “sea”. As we can observe that the data vectors of the class “sea” out numbers the data vector of class “forest” by 191 ($510-319=191$) that would implicitly push away prototypes of the smaller “forest”

class, hence we have to equalize the data vectors of both classes.

To do data vector equalization in the training set, we took the “forest” training set, randomly selected 191 data vectors from it, made copies of these data vectors and added them back in “forest” training set that increased the size of “forest” training set from 319 to 510 which is comparable to the “sea” training set.

We have initiated the algorithm with one prototype for each class (fig 6.1.b) with **learning rate wrong** $\varepsilon^- = 0.001$ (this is the rate by which a prototype will be pushed away from a rival class point, so blue points will push red prototypes away with this rate from them and vice versa), **learning rate correct** $\varepsilon^+ = 0.01$ (this is the rate by which a prototype will be attracted towards points of its class, so blue points attract the blue prototypes toward them and vice versa, please note that the attraction and repulsion of points and prototypes depends upon the distance between them i.e. points far away from prototype will have less influence compared to the point nearer to it) and **learning rate lambda** $\varepsilon = 0.001$ (this is the rate by which the relevance of X and Y coordinates will be learned). After that, we run the DGRLVQ algorithm assigning 1000 cycles for a single run (1-Run). The algorithm found the solution model by adding dynamically 3 prototypes of class “forest” and 3 prototypes of class “sea” (fig 6.1.c). We repeated our experiment 5000 times (5000-Run) with the same configurations but with different datasets i.e. we generated training and testing datasets by picking examples on a random before each run. The DGRLVQ obtain an accuracy of **100%**.

Geographical satellite dataset

Dimension=2 Classes=2 Samples=3315 Data-PreProcessing=None

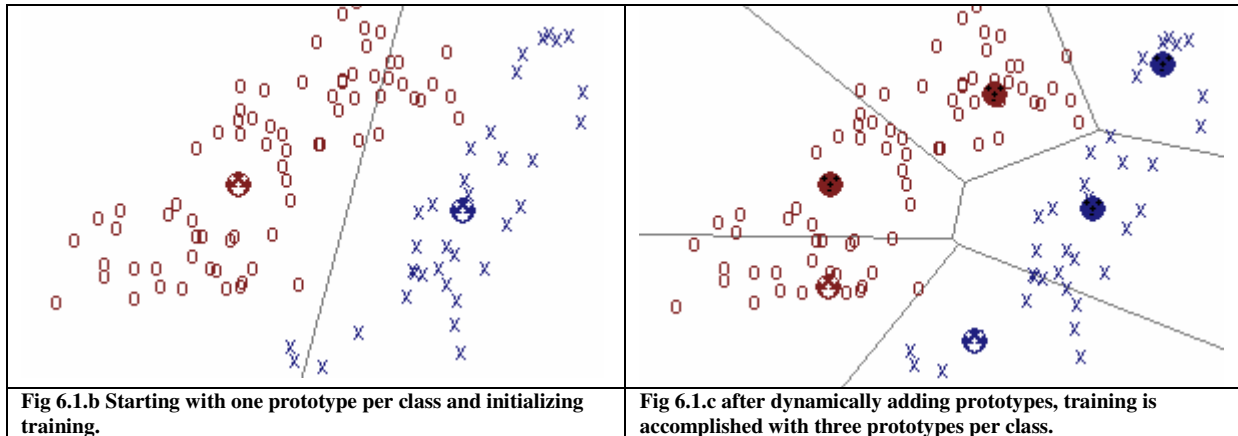
T%	PCT	% Corr.	% Classif.	Sensitivity	Specificity	PPV
25	1.0	99.52	100.00	99.43	99.76	99.45
30	1.0	100.00	100.00	100.00	100.00	100.00

T%: training percent; PCT: probability classification threshold; %Corr: percent correctly classified;
%Classif: percent classified; PPV: positive predictive value

Other Experimental information

Relevance Factor above 0.01 threshold (value: dimension)	0.49 : 0.51 : 1
Learning Rate Wrong	$\varepsilon^- = 0.001$
Learning Rate Correct	$\varepsilon^+ = 0.01$
Learning Rate Lambda	$\varepsilon = 0.001$
Classification Type	Binary, two class
Updates per run	1000
Lambda normalization	True
Exponent of lambda	2
Exponent of distance metric	2
Dynamic shrinking and expansion of network	ON

Learning decay	0%
Qualifying winner count threshold	OFF
Prototype Allocation method	Mean method
Prototype addition method	Automatic
Prototype distribution(class: number of prototypes)	C0:3, C1:3



6.2 Iris

In a second test we applied DGRLVQ to the well known Iris data set provided in the UCI repository of machine learning [6]. The task is to predict three classes of plants based on 4 numerical attributes in 150 instances, i.e., we deal with data points in \mathbb{R}^4 with labels in $\{0,1,2\}$

The dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other. The task is to predict three classes of plants based on numerical attributes in 150 instances. Please note that the Sensitivity and Specificity is applicable only for binary classifications. We repeated our experiment 2000 times (2000-Run)

Iris Dataset

Dimension=4 Classes=3 Samples=150 Data-PreProcessing=None

T%	PCT	% Corr.	% Classif.	Sensitivity	Specificity	PPV
75	1.0	96.42	100.00	NA	NA	NA
95	1.0	97.68	100.00	NA	NA	NA

T%: training percent; PCT: probability classification threshold; %Corr: percent correctly classified; %Classif: percent classified; PPV: positive predictive value; NA: Not Applicable

Other Experimental information

Relevance Factor above 0.01 threshold (value: dimension)	0.039:0 0.044:1 0.85:2 0.061:3
Learning Rate Wrong	$\varepsilon^- = 0.00001$
Learning Rate Correct	$\varepsilon^+ = 0.01$
Learning Rate Lambda	$\varepsilon = 0.00001$
Classification Type	3 class
Updates per run	5000
Lambda normalization	True
Exponent of lambda	2
Exponent of distance metric	2
Dynamic shrinking and expansion of network	ON
Learning decay	0%
Qualifying winner count threshold	OFF
Prototype Allocation method	Mean method
Prototype addition method	Automatic
Prototype distribution(class: number of prototypes)	C0:1, C1:4, C2:3

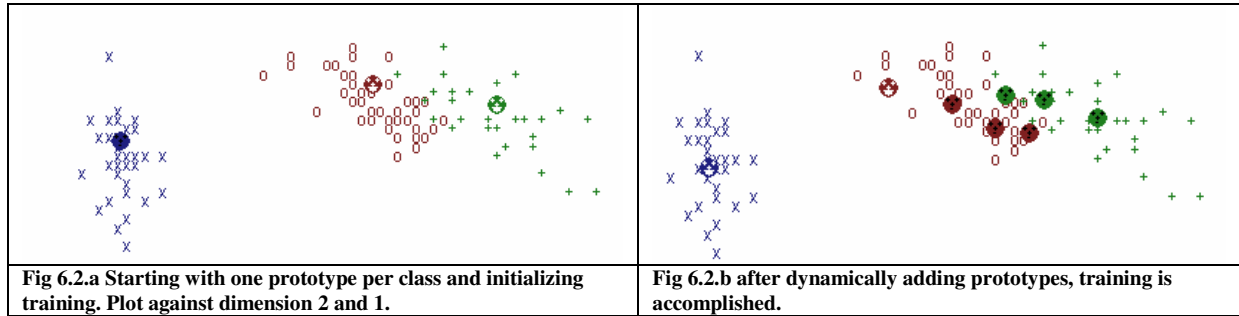
Both, LVQ and RLVQ obtain an accuracy of about 95% for a training and test set if trained with 2 prototypes for each class. RLVQ shows a slightly cyclic behaviour in the limit, the accuracy changing between 94% and 96%. GRLVQ yields the better accuracy of at least 96% on the training as well as the test set.

As compared with other algorithms, DGRLVQ performs better. The following table shows that DGRLVQ achieves promising results.

Comparison Table

Algorithm	LVQ	RLVQ	GRLVQ	DGRLVQ
Mean % Accuracy	94	95	96	97.68

Please note that a better accuracy of about 100% would be possible as reported in the literature. We could not produce such a solution with DGRLVQ. Moreover, a perfect recognition of 100% would correspond to overfitting since the data comprises small noise as reported in the literature.



Note: Iris is 4 dimensional data and we have plotted above a scatter plot with second dimension (x-axis) against first dimension (y-axis).

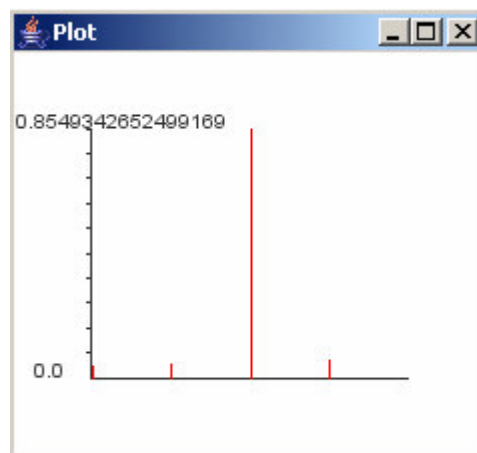


Fig 6.2.c Plot showing the relevance of third dimension is higher than the other dimensions.

6.3 Mushroom

This data set [6] includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the *Agaricus* and *Lepiota* Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like "leaflets three, let it be" for Poisonous Oak and Ivy. The task is to predict if a given mushroom is eatable or poisonous depending upon 22 symbolic attributes. Unary encoding of 22 attributes lead to 8124 patterns with dimension of 117. We repeated our experiment 1000 times (1000-Run).

Mushroom Dataset**Dimension=117 Classes=2 Samples=8124 Data-PreProcessing=None**

T%	PCT	% Corr.	% Classif.	Sensitivity	Specificity	PPV
75	1.0	98.88	100.00	98.90	99.01	98.70
95	1.0	99.98	100.00	99.78	99.67	99.70

T%: training percent; PCT: probability classification threshold; %Corr: percent correctly classified;
 %Classif: percent classified; PPV: positive predictive value

Other Experimental information

Relevance Factor above 0.01 threshold (value: dimension)	0.0174 : 26 0.0185 : 27 0.0121 : 33 0.0124 : 34 0.0141 : 35 0.0138 : 36 0.0140 : 40 0.0119 : 49 0.0121 : 50 0.0136 : 53 0.0128 : 57 0.0106 : 59 0.0118 : 60 0.0128 : 72 0.0112 : 81 0.0119 : 92 0.0111 : 93 0.0110 : 94 0.0107 : 100 0.0103 : 106
Learning Rate Wrong	$\varepsilon^- = 0.001$
Learning Rate Correct	$\varepsilon^+ = 0.01$
Learning Rate Lambda	$\varepsilon = 0.00001$
Classification Type	2 class
Updates per run	5000
Lambda normalization	True
Exponent of lambda	2
Exponent of distance metric	2
Dynamic shrinking and expansion of network	ON
Learning decay	0%
Qualifying winner count threshold	OFF
Prototype Allocation method	Mean method
Prototype addition method	Automatic
Prototype distribution(class: number of prototypes)	C0:9, C1:8

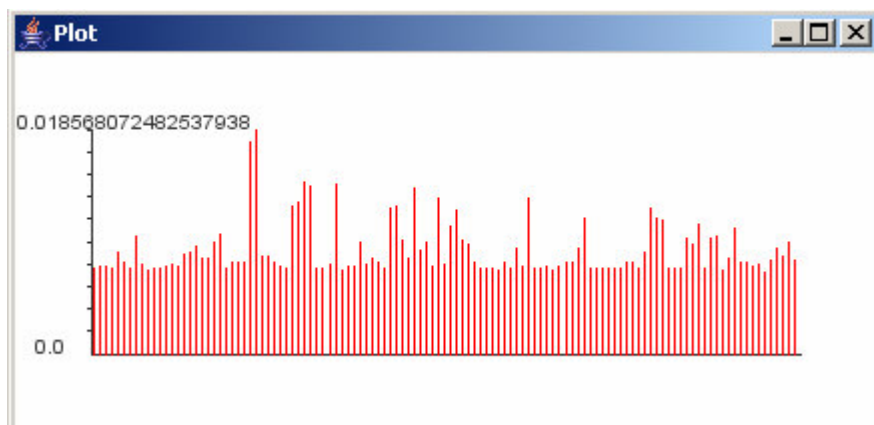


Fig 6.3.a Plot showing the relevance of the dimensions

Comparison Table

Algorithm	SRNG	BBTreeSRNG	DGRLVQ
Mean % Accuracy	97.5	98.7	99.98

6.4 Ovarian Cancer

The use of Mass Spectrometry (MS) for determining the masses of biomolecules and biomolecular fragments present in a complex sample mixture is an emerging robust technique for cancer detection. Mass Spectrometry provides ultra-high resolution mass information as a form of a mass spectrum consisting of a set of m/z values and corresponding relative intensities that are a function of all ionized molecules present with that m/z ratio. The mass spectrum observed for a sample is thus a function of the molecules present. Experimental conditions that affect the molecular composition of a sample should therefore affect its mass spectrum. Mass spectrometry is therefore often used to test for the presence or absence of one or more molecules. The presence of such molecules may indicate a particular enzymatic activity, disease state, cell type, or condition.

If each spectrum is sampled at the same m/z values then we can represent each spectrum as a point in an r -dimensional space, where r is the number of m/z values for which relative intensities are recorded per spectrum. We call this space spectral-space. Each spectrum is therefore represented in spectral-space by the point $(p_1 \dots p_r)$, where p_i is the relative intensity observed at the i th m/z value. Similar spectra will inherently cluster in spectral-space. The assumption made is that in spectral-space, healthy spectra form one cluster while disease spectra form a second, nonoverlapping cluster. The hypothesis for classification is that any healthy spectrum lies closer to the healthy cluster than to the disease cluster (and vice-versa). Unclassified spectra can then be classified by assigning them to their nearest cluster [21].

Datasets were obtained from the NIH and FDA Clinical Proteomics Program Databank [13]. The healthy samples in the ovarian cancer datasets come from women at risk for ovarian cancer (the demographic most likely to use and benefit from serum screening) while the ovarian cancer positive samples come from women with tumours spanning all major epithelial subtypes and stages of disease.

We performed no preprocessing on the datasets and we tested DGRLVQ algorithm directly on the raw dataset as it was obtained. We repeated our experiment 1000 times (1000-Run) and found that the accuracy of about 99.98% can be achieved. Please note that DGRLVQ performance is comparably good even with 5% training split and data dimension of 15154.

Ovarian Cancer Dataset

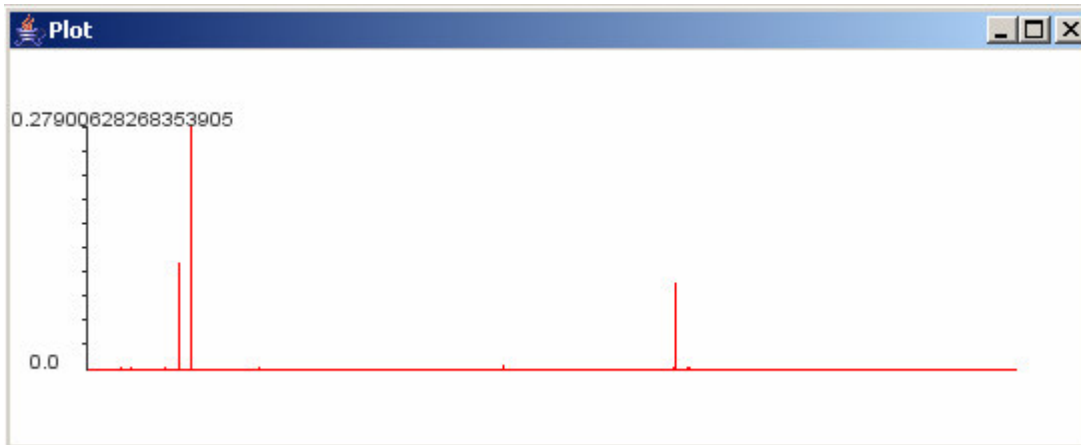
Dimension=15154 Classes=2 Samples=253 Data-PreProcessing=None

T%	PCT	% Corr	% Classif.	Sensitivity	Specificity	PPV
5	1.0	97.45	100.00	97.33	96.89	97.35
95	1.0	99.98	100.00	100.00	99.99	99.99

T%: training percent; PCT: probability classification threshold; %Corr: percent correctly classified; % Classif: percent classified; PPV: positive predictive value

Other Experimental information

Relevance Factor above 0.01 threshold (value: dimension)	0.0408 : 1490 0.1163 : 1491 0.1216 : 1494 0.1324 : 1678 0.2790 : 1679 0.0257 : 9573 0.0759 : 9574 0.0985 : 9575 0.0621 : 9576 0.0232 : 9577
Learning Rate Wrong	$\varepsilon^- = 0.000001$
Learning Rate Correct	$\varepsilon^+ = 0.00001$
Learning Rate Lambda	$\varepsilon = 0.000001$
Classification Type	2 class
Updates per run	500
Lambda normalization	True
Exponent of lambda	2
Exponent of distance metric	2
Dynamic shrinking and expansion of network	ON
Learning decay	0%
Qualifying winner count threshold	OFF
Prototype Allocation method	Mean method
Prototype addition method	Automatic
Prototype distribution(class: number of prototypes)	C0:5, C1:5



Plot showing the relevance of the dimensions

Comparison Table

Algorithm	Q5	DGRLVQ
Mean % Accuracy	100	99.98

Please note that Q5 [21] is an exact algorithm whose working is based upon Principal Component analysis (PCA), Linear discriminant analysis and Back-Projection on the dataset whereas, DGRLVQ a heuristic classification algorithm that does not apply any preprocessing methods to the dataset.

6.5 Prostate Cancer

The dataset is obtained by spectral analysis of blood plasma of patients suffering from cancer and probands. The dataset for prostate cancer [12] [17] detection consists of 329 spectra disease sample and 316 spectra from control samples. The disease samples contain three classes of different diseases (BPH, CAB & CCD). In the experiments, each disease class is compared to the class with spectra from the control group. The mass range of 2 to 20KDa was used. The preprocessed data has been provided by F. Schleif. The spectra were first processed using the following standardized workflow.

- Baseline subtraction and peak detection
- Normalization of all spectra to their own total ion count (TIC)
- Recalibration of spectra on each other using the most prominent peaks
- Definition of mass ranges which were considered as peaks
- Calculation of peak area for each spectrum
- Auto-scaling of peak area to get the zero mean and unit variance

We repeated our experiment 1000 times (1000-Run) and found that the following results

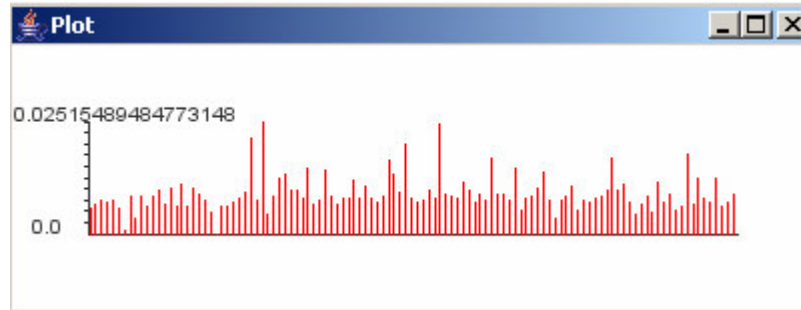
Prostate Cancer Dataset (CCD vs. Control)
Dimension=114 Classes=2 Samples=326 Data-PreProcessing=Yes

T%	PCT	%Corr	%Classif.	Sensitivity	Specificity	PPV
25	1.0	96.15	100.00	96.34	97.69	96.45
95	1.0	98.58	100.00	99.10	98.09	98.49

T%: training percent; PCT: probability classification threshold; %Corr: percent correctly classified;
 %Classif: percent classified; PPV: positive predictive value

Other Experimental information

Relevance Factor above 0.01 threshold (value: dimension)	0.0103 : 14 0.0110 : 16 0.0214 : 28 0.0251 : 30 0.0123 : 33 0.0131 : 34 0.0147 : 38 0.0141 : 41 0.0117 : 46 0.0106 : 48 0.0164 : 52 0.0131 : 53 0.0198 : 55 0.0245 : 61 0.0113 : 65 0.0166 : 70 0.0147 : 74 0.0102 : 78 0.0136 : 79 0.0103 : 84 0.0167 : 91 0.0110 : 93 0.0114 : 99 0.0178 : 104
Learning Rate Wrong	$\varepsilon^- = 0.000001$
Learning Rate Correct	$\varepsilon^+ = 0.025$
Learning Rate Lambda	$\varepsilon = 0.002$
Classification Type	2 class
Updates per run	350
Lambda normalization	True
Exponent of lambda	2
Exponent of distance metric	2
Dynamic shrinking and expansion of network	ON
Learning decay	0%
Qualifying winner count threshold	OFF
Prototype Allocation method	Mean method
Prototype addition method	Automatic
Prototype distribution(class: number of prototypes)	C0:5, C1:5



Plot showing the relevance of the dimensions

Prostate Cancer Dataset (BPH vs. Control)**Dimension=104 Classes=2 Samples=316 Data-PreProcessing=Yes**

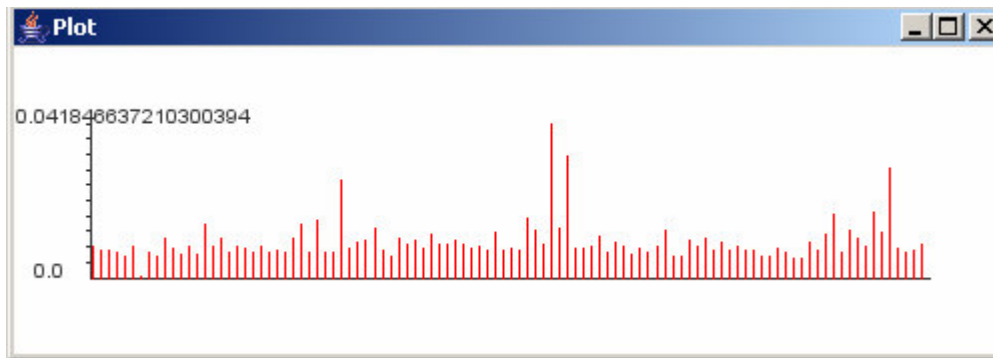
T%	PCT	% Corr	% Classif.	Sensitivity	Specificity	PPV
25	1.0	91.55	100.00	91.24	93.69	92.52
95	1.0	94.81	100.00	94.11	95.09	94.19

T%: training percent; PCT: probability classification threshold; %Corr: percent correctly classified;
 %Classif: percent classified; PPV: positive predictive value

Other Experimental information

Relevance Factor above 0.01 threshold (value: dimension)	0.0105 : 9 0.0145 : 14 0.0104 : 16 0.0105 : 25 0.0144 : 26 0.0152 : 28 0.0261 : 31 0.0132 : 35 0.0105 : 38 0.0116 : 42 0.0121 : 50 0.0157 : 54 0.0125 : 55 0.0418 : 57 0.0132 : 58 0.0331 : 59 0.0110 : 63 0.0128 : 71 0.0103 : 74 0.0104 : 76 0.0117 : 91 0.0171 : 92 0.0126 : 94 0.0105 : 95 0.0178 : 97 0.0124 : 98 0.0295 : 99
Learning Rate Wrong	$\varepsilon^- = 0.000001$
Learning Rate Correct	$\varepsilon^+ = 0.025$
Learning Rate Lambda	$\varepsilon = 0.002$
Classification Type	2 class
Updates per run	350
Lambda normalization	True
Exponent of lambda	2
Exponent of distance metric	2
Dynamic shrinking and expansion of network	ON
Learning decay	0%
Qualifying winner	OFF

count threshold	
Prototype Allocation method	Mean method
Prototype addition method	Automatic
Prototype distribution(class: number of prototypes)	C0:5, C1:5



Plot showing the relevance of the dimensions

Prostate Cancer Dataset (CAB vs. Control)

Dimension=111 Classes=2 Samples=325 Data-PreProcessing=Yes

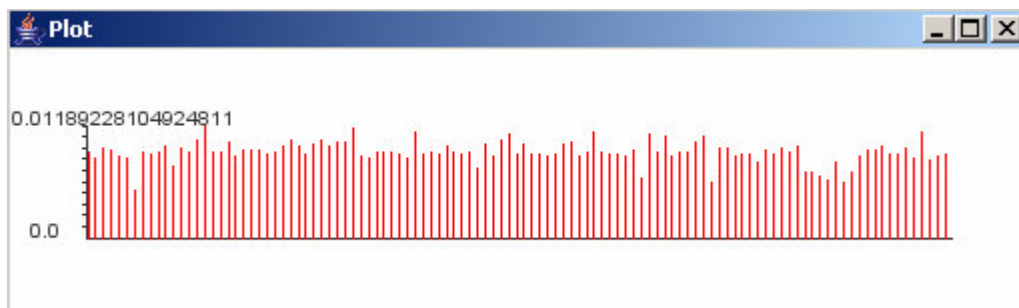
T%	PCT	% Corr	% Classif.	Sensitivity	Specificity	PPV
25	1.0	93.35	100.00	94.54	93.99	94.13
95	1.0	95.13	100.00	95.63	94.24	95.14

T%: training percent; PCT: probability classification threshold; %Corr: percent correctly classified;
 %Classif: percent classified; PPV: positive predictive value

Other Experimental information

Relevance Factor above 0.01 threshold (value: dimension)	0.0103 : 14 0.0118 : 15 0.0100 : 18 0.0103 : 26 0.0102 : 30 0.0100 : 32 0.0100 : 33 0.0115 : 34 0.0110 : 42 0.0103 : 53 0.0109 : 54 0.0100 : 62 0.0112 : 65 0.0110 : 72 0.0107 : 74 0.0101 : 78 0.0107 : 79 0.0110 : 107
Learning Rate Wrong	$\varepsilon^- = 0.000001$
Learning Rate Correct	$\varepsilon^+ = 0.025$
Learning Rate Lambda	$\varepsilon = 0.002$
Classification Type	2 class
Updates per run	350
Lambda normalization	True

Exponent of lambda	2
Exponent of distance metric	2
Dynamic shrinking and expansion of network	ON
Learning decay	0%
Qualifying winner count threshold	OFF
Prototype Allocation method	Mean method
Prototype addition method	Automatic
Prototype distribution(class: number of prototypes)	C0:5, C1:5



Plot showing the relevance of the dimensions

The following tabular shows that our results are competitive to the results from [17]. Thereby, GRLVQ is a version of prototype based clustering without automatic determination of prototypes, thus more complex to use. UMSA is a variant of SVM. Thus results achieved in this thesis can be seen as promising.

Comparison Table

Dataset	UMSA	GRLVQ	DGRLVQ
CCD vs. Control	97.38	98.69	98.58
BPH vs. Control	93.52	94.80	94.81
CAB vs. Control	94.15	95.04	95.13

Conclusions and Discussion

In most real world problems, we face complex data with high dimensionality and the data availability is also limited. Manual analysis of such kind of data is impractical and so these problems led to the development of different kinds of classification algorithms. Some solutions are based upon exact statical algorithms those may be fit for one kind of situations but are not flexible for others. Different algorithms generate a solution model effectively but only for complexity reduced data by applying dimensionality reduction methods (PCA) before an analysis. Usually, the performance of algorithms is inversely proportional to the data dimension and directly proportional to the number of available data. One of the known issues specifically related to biological data is the large number of input versus the small number of available samples. For example, data spectra from mass spectrometry for cancer detection have very high dimension (15000-20000) but the available number of cancer patients is limited. The GRLVQ algorithm is a prototype based method that has superb capability to extract features from multi modal data and tries to develop a data model based upon the relevance of the dimension but this setting has several drawbacks. One drawback is that the number of prototypes per class is to be defined beforehand. In addition, the algorithm is very sensitive to initialization of prototypes, since the cost function is highly multimodal. If a prototype is for some reason ‘outside’ the cluster which it should represent and if there are points of a different class in between, then the other points act as a barrier and the prototype will not find its optimum position. Efforts have been made to introduce a neural gas dynamic for this reason, which ensures that all prototypes spread faithfully among the classes but this is rather a costly approach.

In this thesis, we have presented the DGRLVQ algorithm and have demonstrated its proficiency in the presence of high dimensional complex data with a limited number of available data samples. The performance of DGRLVQ remains consistent even in the presence of very few training examples and the data model generated by DGRLVQ is highly resistant to noise. The DGRLVQ clustering algorithm is a new robust method for dynamically addition and removal of prototypes. DGRLVQ is independent of the initial number of prototypes and the network size shrinks or increases depending upon decision made by algorithm itself. DGRLVQ was successfully tested on artificial as well as real world data, producing promising results as compared to other approaches. DGRLVQ is a quit fast algorithm as it adds prototypes dynamically and that reduces the overheads as compared to GRLVQ where the number of prototypes is predefined. We have also embedded the probabilistic classification model within the algorithm. The use of probability classification framework increases the predictive accuracy of DGRLVQ. The tradeoff is represented between confidence in classification and number of samples classified. Our results show that a classification threshold can be chosen for DGRLVQ such that the samples are classified

with sensitivity, specificity and PPV near 100%. The consistent high level of performance for different testing splits shows that DGRLVQ does not overfits to the training samples. The model generalization is an inherited property of DGRLVQ algorithm from GRLVQ and relevance property of DGRLVQ can efficiently identifies the novel biomarkers or best relevant features. DGRLVQ is not constraint to only binary classification rather it can be used for the parallel classification of unlimited number of classes that adds to the classification speed of algorithm. The classifier generated by DGRLVQ is small and compact because it is made up of only prototype coordinates and relevance vector functions.

Finally, we have noted that DGRLVQ has good performance compared to other algorithms when 1) data has high dimension 2) data is multiclass 3) data has number of relevance features; these are typical situations for data mining in Bioinformatics.

References

- [1] Teuvo Kohonen, Self-Organizing Maps, vol. 30 of Springer Series in Information Sciences, Springer, Berlin, Heidelberg, 1995 (Second extended edition 1997)
- [2] A.S. Sato and K. Yamada, "Generalized learning vector quantization" in Advances in Neural Information Processing Systems, G. Tesauro, D. Touretzky, and T. Leen, Eds., vol 7 pp. 423-429. MIT Press, 1995
- [3] B. Hammer and Th. Villmann, "Generalized relevance learning vector quantization" Neural Networks, vol. 15, no. 8-9, pp. 1059-1068, 2002
- [4] B.Hammer, M. Strickert, and Th. Villmann, "Learning vector quantization for multimodal data" in Proc. International Conf. on artificial Neural Networks (ICANN), J.R. Dorronosoro, Ed., LectureNotes in Computer Science 2415, pp. 370-376. Springer Verlag, 2002
- [5] Th. Villman, F.M. Schleif and B.Hammer, "Supervised Neural gas & relevance learning in LVQ"
- [6] C. Blake and C.Merz UCI Repository of machine learning databases, 1998
- [7] Bishop, C.M. (1995), Neural Networks for Pattern Recognition, Oxford: Oxford University Press
- [8] Russell C. Eberhart, Roy W. Dobbins, Neural network PC Tools, Academic Press, 1990
- [9] Abhijit S. Pandya, Robert B. Macy, Pattern Recognition with Neural Networks in C++, IEEE Press, 1996
- [10] Sarle, W.S., ed. (1997), Neural Network FAQ, part 1 of 7: Introduction, periodic posting to the Usenet newsgroup comp.ai.neural-nets, URL: <ftp://ftp.sas.com/pub/neural/FAQ.html>
- [11] Haykin, S. (1994), Neural Networks: A Comprehensive Foundation, NY: Macmillan.
- [12] Eastern Virginia Medical School - The Virginia Prostate Center.
<http://www.evms.edu/vpc/seldi/index.html>, 2002.
- [13] NIH and FDA Clinical Proteomics Program Databank. <http://clinicalproteomics.steem.com>, 2002.
- [14] BL. Adam, Y. Qu, J. Davis, M.Ward, M. Clements, L. Cazares, OJ. Semmes, P. Schellhammer, Y. Yasui, Z. Feng, and G.Wright Jr. Serum protein fingerprinting coupled with a pattern-matching algorithm distinguishes prostate cancer from benign prostate hyperplasia and healthy men. Cancer Research, 62:3609–3614, 2002.
- [15] B. Austen, E. Frears, and H. Davies. The use of SELDI proteinchip arrays to monitor production of Alzheimer's β -amyloid in transfected cells. Journal of Peptide Science, 6:459–469, 2000.
- [16] H.M.Heise, S.Haiber, M.Licht, D.F.Ihrig, C.Moll, M.Stuecker, In-vivo measurements of intact human skin by near- and mid-infrared spectroscopy, to appear in VDI Forschungsberichte
- [17] F. Schleif, Bruker Daltonics and Thomas Villmann, University Leipzig, Personal Communication.

- [18] <http://thefreedictionary.com/> Farlex, Inc. 347 Keats Rd. Huntingdon Valley, PA 19006, USA
- [19] Mark Chatterley and Tom Radcliffe, 2003, GeneLinker Gold (or Platinum) 4.0, Kingston, Predictive Patterns <http://www.predictivepatterns.com/>
- [20] An introduction to support vector machines (and other kernel-based learning methods) N. Cristianini and J. Shawe-Taylor Cambridge University Press 2000
- [21] Probabilistic Disease Classification of Expression-Dependent Proteomic Data from Mass Spectrometry of Human Serum, Ryan H. Lilien, Hany Farid and Bruce R. Donald, Journal of Computational Biology, 10(6):925-946 (2003)

Herewith I affirm that I have written the Thesis independently. I have used only the sources and aids mentioned in the Thesis.

Osnabrück, Tuesday, November 23, 2004

Signature