CHALMERS

A Cluster Analysis Framework using DiMaxL

Clustering and data reduction in the presence of noise

Master of Science Thesis in the Programme Computer Science:

Algorithms, Languanges and Logic

PHILIP WERNER

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, September 2010

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.
The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

A Cluster Analysis Framework using DiMaxL
Clustering and data reduction in the presence of noise

PHILIP D. E. F. WERNER

© PHILIP D. E. F. WERNER, September 2010.

Examiner: PETER DAMASCHKE

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden September 2010

Computer Science and Engineering
Göteborg 2010

Abstract

The goal of this project was to develop a framework which can be used to make accurate predictions on large, sampled, data sets where statistical outliers are present. A secondary aim was to develop a method to reduce the large amount of data sometimes available, but all of it not always useful when making a prediction. The framework was developed using the DiMaxL algorithms and was tested on data sets taken from biology. These data sets are protein measurements where a large amount of statistical outliers are present. The results indicate that the method can accurately detect patterns even in presence of large amount of noise without any excessive overfitting. In the case of data reduction, the accuracy of the method is more sensitive to the amount of available data, and a semi-automatic procedure is recommended. In conclusion, the framework developed, is able to effectively remove noise while detecting the underlying pattern present, even in complex correlations.

"*Simplicity is the ultimate sophistication*"
- Leonardo da Vinci

Acknowledgments

I would like to thank my supervisor Peter Damaschke for giving me the opportunity to work with such an important project, and last, but not least, for his patience.

Contents

1 Introduction

In many areas of application, such as the life sciences, large collections of sampled data need to be organized and structured. One way of doing this is through a process called cluster analysis, or clustering for short. Once this process is complete, the structured data can be used to make predictions on future instances. One short-coming of the clustering techniques in use today is the inability, or poor performance, in situation where statistical outliers are present in the data sets, and these are not uncommon in the setting described above. A method thus needs to be developed which can operate in a fast manner, as the number of instances are large, while being able to filter the statistical outliers.

In some cases a large collection of data is sampled while only a subset is needed to make an accurate prediction. By reducing the amount of data the complexity of the problem is reduced, while also, in some cases, increasing the accuracy of the prediction. A secondary problem is thus to find which data is of importance to the predictions one would like to make.

These two problems are important to solve in cases where other traditional machine learning methods have difficulty achieving high accuracy in their predictions.

The cluster analysis framework presented in this paper is meant to solve the questions stated above. The results indicate that even with data sets taken from biology, where the amount of statistical outliers is high, the predictions made are accurate. Although reducing the data set to a smaller subset does show promise is suffers from lack of data in some cases. A further study will need to be made to verify the full accuracy of the framework, although early results show promise.

2 Background

Finding patterns in data suggests an underlying correlation between the distributions from which the data was sampled. These patterns can in turn be used to help predict yet seen instances. The interdisciplinary field of machine learning studies how finding, and potentionally learning, these patterns may be done with the aid of computer software. There are mainly two, fundamentally different, methods used when trying to find or learn patterns in data.

Supervised learning aims to learn a mathematical function, known in literature as the objective or target function, based on a set of training examples. Input and output to this function can be continuous (regression), discrete (classification), or a combination of both. The term supervised stems from the fact that these training examples have to be classified prior to learning and thus require intervention from a *supervisor*. One obvious desire when selecting these training examples is to cover a large area of the input domain (with given outputs) in such a way that the function being deduced accurately mimics the underlying correlation of the data. The computational difficulty for these learners is not during classification, but instead when the objective function is being learned.

Some of the more well known methods used to represent and learn these target functions are Artificial Neural Networks (ANN), Decision Trees (DT) and Bayesian Networks (BN). These methods have been successfully applied to problems such as hand-written character [12, 24] and face recognition[26, 23], as well as various expert systems [7, 8, 11].

As the correlation between the distributions become ever more complex, the ability of these methods to mimic this correlation accurately is reduced. One reason why they perform poorly, as the correlation becomes increasingly complex, is that they try to generalize the target function over a large domain. A way to remedy this is to divide the large domain into smaller subsets and then learn these individually, hoping that a smaller domain can be more accurately described by the learner. This type of supervised learner is known as a kernel method and has been applied to problems such as protein folding [21] and text categorization [15].

The second method used is known as *Unsupervised learning* and does not require any training prior to classification or regression. Instead the method seeks to organize the data in some order, which in turn can be used to make predictions. Compared with the supervised approach, this method does not make predictions based on a fixed number of training examples, but instead uses the whole data set when making inference. This can make classification (regression) very accurate if the data set is large enough, but at the expense

of computational complexity. In cases where the complexity of the underlying distribution is too difficult for supervised learners to make accurate predictions, unsupervised learning methods provide a good substitute, albeit its computational demand during classification (regression).

A popular method used in unsupervised learning is that of clustering. This method tries to organize the data into subsets of groups based on a notion of similarity between the objects. A common way to group objects together is by using a distance measure (usually the Euclidean in the continuous case) and those defined as being *near* one another form a cluster. Clustering is closely related to density measuring, as one is trying to find dense regions of data as defined by the distance.

The motivation behind this paper can be found in problems originating from the life sciences where the correlation of the underlying distribution sometimes prove too difficult for supervised learners to perform efficiently (as described above) [1, 22, 6]. Although supervised methods can perform well even in situations where there is noise present, clustering can be more sensitive to this problem. The data taken from, but not limited to, life sciences may be sampled with an instrument using low precision and as such contains a number of statistical outliers that need to be filtered in some way. Instead of trying to improve the ability of supervised learners in this setting, the approach taken in this paper is to use clustering on the full data sets taken from imprecise measurement instruments and overcome the problems related to statistical outliers.

The remainder of this paper is outlined as follows. Section 2.1 introduces the reader to the field of protein structure prediction and the methods from machine learning that try and solve this problem. An integral part of the methods described in the main sections of the paper make use of the DiMaxL algorithms developed by Bergkvist and Damaschke [2] which are described in 2.2. For a more in-depth look at the theory behind the DiMaxL algorithms the reader is directed to [2]. The implementation, analysis and evaluation of the algorithms can be found in [10]. A broad overview of the common clustering techniques in use today follows in section 3. The main section of the report comes after that.

2.1 Protein structure prediction using machine learning

Proteins are groups of organic compound that are *essential* for many of the functions and building blocks of living organisms [5]. Functions where protein plays an important role range from transportation of oxygen through the blood, the cataclysmic reactions triggered by enzymes to hormons acting as messengers between cells, to name a few. The building blocks of nails,

hair, fur and muscles are all built up by proteins. Every living organism contains huge quantities of protein, and these are, to a large extent, integrally connected to the mechanisms of the living cell. Without protein these mechanisms would not be able to function, and no living organism could exist.

The function of a protein is determined by its three-dimensional structure. This structure can be divided into four levels, which represent how the protein folds (evolves) from a small molecular structure to a fully functional protein. By knowing this structure one can determine its functional use.

At the first level, known as the primary structure, a chain of amino acids are connected to each other. An amino acid is a molecule consisting (mainly) of carbon, nitrogen, oxygen and hydrogen atoms. The simplest amino acid contains only eight atoms, while the larger ones have close to thirty. There are around twenty, naturally occuring, amino acids which are vital for the human body to function properly. Ten of these amino acids can be produced naturally in the body, while the others have to be injected by other means, such as food. Figure 1 shows a simple amino acid, while figure 2 shows how two amino acids connect to form a chain, known as the back-bone structure. The number of amino acids forming a back-bone structure range from a few to several thousands, although the more common have between 200-300.

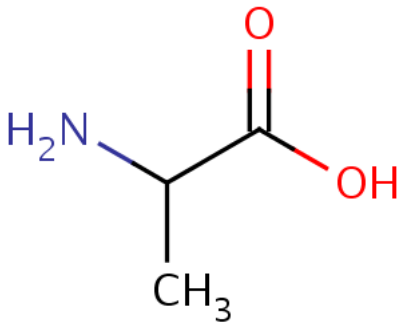

Figure 1: The Alanine amino acid.

The way in which the chain of amino acids connect to one another as a back-bone form the secondary structure. Although there are huge number of possible combinations of back-bone stuctures, the amino acids are restricted by how they can connect to each other. This connection is done by creating a so called peptide bond between the acids, which in turn also releases water during the chemical process. A peptide bond is a molecule consisting of

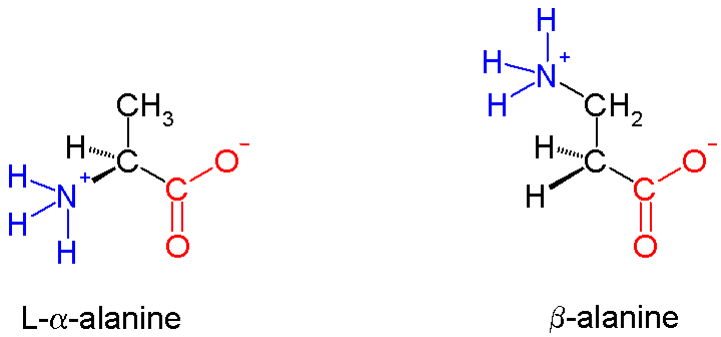

Figure 2: Two Alanine amino acids connected to form a back-bone. The molecules can bond in different ways as can be seen from the two images.

Carbon (C), Oxygen (O), Hydrogen (H) and Nitrogen (N) atoms, as can be seen in figure 3. These atoms create a plane, where no rotation in the molecular structure is possible, thus restricting some parts of the back-bone structure.

There are, however, other areas where rotation is possible. The bonds between the nitrogen and carbon atom, as well as the bond between the two carbon atoms, as is illustrated in 3, are both single bonds, and thus allow for rotation to happen. These angles, denoted by the greek letters ϕ and ψ, ultimately decide the shape of the secondary structure. By learning the mechanism which controls the rotation, the hope is that one may be able to understand parts of the folding process. This mechanism is, however, none-trivial and has been studied extensively in literature with varying success [6, 22, 1].

Instead of trying to learn how the folding mechanism works, there are empirical methods which try to solve parts of the problem by using a more direct approach. One method, pioneered by the K. Wüthrich [25], was to use nuclear magnetic resonance (NMR) to determine the molecular structure of protein. NMR measures the resonance (atomic spin) of a nucleus in a strong magnetic field and this value is known as the *chemical shift*. This chemical shift is sensitive to the molecular structure surrounding the nucleus, and can thus be used to determine the environment around it. By using the correlation between chemical shift values and measured torsion angles, the hope is to be able to restrict the torsion angles depending on the value of the chemical shift for specific nuclei. One can then create inference rules from these restrictions, which in turn can be used by computer software to help understand the complex folding process.

One advantage of NMR is the vaste amount of measurements available

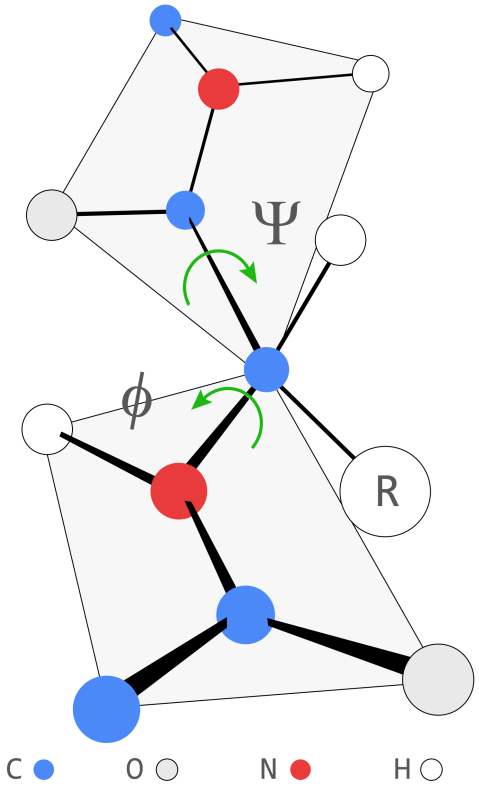

Figure 3: An image showing a peptide bond and the corresponding planes constructed. The rotation of the angles ϕ and ψ determines the secondary structure.

from public protein data banks. A downside, however, is the high amount of noise present in these measurements that need to be filtered for the predictions to be accurate.

2.2 The DiMaxL Algorithms

The problem described in the previous section motivated the development of the DiMaxL algorithms. More precisely, the creators of the algorithms, Bergqvist and Damaschke, sought to use the NMR measurements to try and predict the torsion angles, but in doing so needed an efficient method to remove noise.

In the case of two dimensions, this problem can be viewed as trying to find the largest rectangles containing a few sporadic points (noise). This method of finding the empty regions in data, known as anti-clusters, is an efficient

complement to the more commonly used method of finding the dense regions. But even in the modest case just explained, there is no known polynomial time algorithm [2]. As such, they simplified the problem to one dimension and created two algorithms designed to solve the problem on the line, both efficiently and optimally. The algorithms find the optimal subsequences (intervals) on the line containing a few sporadic points. This solves one part of the problem, although the multi-dimensional case still remains to be solved.

The idea put forward in their paper, to solve the multi-dimensional case, was to project a multi-dimensional window onto one dimension, and then run the algorithms to find the large intervals on the line. This simplification is based on the two following assumptions. The first being that the distributions in the data set exhibit a natural smoothness, and the second being that the DiMaxL algorithms will return *similar* solutions as the pattern in the windows projected has clearly emerged. This last assumption assumes a method exists which can classify similar DiMaxL solutions to a high degree of accuracy, and in turn is the main motivation behind this paper.

3 Related Work

3.1 Partitioning algorithms

Partition based clustering algorithms divide a number of objects into a predefined number of clusters. The *k-means* clustering algorithm is one of the simplest and oldest algorithms for partitioning a set of objects into K different clusters. It was first described by MacQueen [18] in 1967 and has been used and studied extensively ever since. A search for *k-means* and *clustering* on a popular database returns close to four thousand papers between the years 1990-2010.

Prior to stating how the algorithm computes these K clusters, a set of notations are in order. The N objects $x_1, x_2, ..., x_n$ to partition belong to a D-dimensional standard Euclidean space. Connected with these K clusters is another set of D-dimensional vectors c_k, $k = 1, .., K$, representing the centers at each iteration of the K clusters. The idea behind the algorithm is to connect each object with the closest cluster in such a way that an objective function, describing the squared distance for each point to its connected center point, is minimized [3]. This objective function is shown below. If a point is connected to a cluster c_k then $r_{nk} = 1$ otherwise $r_{nk} = 0$.

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} |x_n - c_k|^2 \tag{1}$$

The algorithm starts by selecting K random points to represent the centers, c_k, for each cluster. During each iteration of the algorithm the following is done: First optimize J by fixing the centers and selecting for each point the cluster which minimizes the distance to its center. Secondly, fix the points and select the point for each c_k which minimizes J. This two-step process is repeated until convergence. The process is guarenteed to converge, but may do so in a local minima, depending on the selection of the initial cluster centers. The clusters created are also guarenteed to be disjoint, as the points can only be connected to one cluster. Due to the fact that the clusters are created by means of a center, the shape of these are spherical (convex).

The original paper describing the algorithm can be found in [18], while a more up-to-date description, including variations of it, is detailed in [3] and [19].

Because of how the choice of center point is selected, the k-means algorithm is not very robust to statistical outliers [4]. If a cluster contains a number of statistical outliers, these will heavily affect the Euclidean distance function when computing the mean centers at every iteration. To solve this

problem a new way of computing the center point, less sensitive to outliers, had to be created. One way of doing this is by selecting the point which has the least cumulative distance to all other points. Although a lot more computationally complex, it is more robust to outliers than the original method. These methods, of selecting the centers in a more robust way, are known as the *medoid* algorithms [17].

A more recent algorithm, CLARANS, tries to overcome the problem of computational complexity by randomized search [20]. Instead of looking at all points in the data set, CLARANS minimizes the search by sampling a small subset of the points. The hope is that a large enough sample will be able to represent, to a large extent, the original data set, but without the overhead of looking at every single point.

Two parameters control the output, as well as the run-time complexity of the algorithm. The parameter *maxneighbour* controls how many neighbours the algorithm should look at during each iteration. By setting this value high, the algorithm effectively works the same way as the original k-medoid algorithm, and thus increases the run-time complexity as well. The second parameter controls the number of iterations the algorithm should run, when looking for the optimal node to select as the cluster center (*medoid*). Obviously the algorithm needs to be run a number of times, depending on the fixed number of clusters to look for.

3.2 Density based algorithms

Instead of allowing all objects to be apart of some cluster, density based clustering makes a more explicit definition of it. These types of algorithms rely on a number of parameters as they find the clusters, most of which need to be tuned for each specific data set. Once these are tuned, however, statistical outliers can sometimes be efficiently removed.

A popular method is to fix K number of points to include in a cluster, and then find the volume around each specific data point where K objects are contained. The *volume* around a data point is defined as a hyper-cube (or some other geometerical shape), and will expand until K objects have been found. The choice of K is critical to finding the true clusters though. A small value of K and underfitting will occur, while a too large value will produce overfitting. This method is referred to as *k-nearest neighbour*.

Another approach is to fix the size of the volume instead of the number of data points. The problem in this case is to choose the size of the window to adequently represent the underlying distribution. As with the k-nearest neighbour approach, different values of this volume will lead to either under- or over-fitting. A second issue relates to differing densities in regions, which

may require a variable approach to selecting the volume. These types of methods are known as *kernel density estimators*.

One advantage of using these approaches, compared to the partitioning methods, is the ability to detect arbitrarily shaped patterns. The disadvantage is obviously to try and determine the value for K or the proper volume.

A well-known density based algorithm is DBSCAN [9]. This algorithm first finds a dense region, which is classified as a cluster, and then expands this region until no other points can be added based on a certain criteria. A region is classified as dense by using two parameters which remain global throughout the search space. The first defines the area to look at around each point, while the second specifies the minimum number of points in this area. If these two criteria are not fulfilled, the point is classified as noise. A point being on the fringe of a cluster may thus be classified as noise at first, until it is *consumed* during later iterations, by a region classified as a cluster. Two clusters can also be merged into one, if they are nearby.

Another algorithm, named DENCLUE, outperforms the run-time factor of DBSCAN by a factor of 45 [13, 14]. The motivation behind the conception of DENCLUE was to be able to cluster high-dimensional data sets where a lot of noise is present. Most other previous algorithms would either be good at detecting clusters in presence of noise, but not be able to scale when the dimensions increased, or vice versa. The algorithm is based on a strong mathematical foundation, where each point is represented by a local density function affecting the neighbourhood around the point. The neighbourhood region is the volume of the box, as described above, and as such determines the *smoothness* of the cluster.

The algorithm works in two steps. The first step determines which regions in space that are populated, and these are stored in some efficient data structure, such as a binary tree. During the second step, the clustering takes place, which only operates on the regions found in the first phase. If a region contains many points, the local density functions representing these will affect this neighbourhood to a large extent and thus be classified as a cluster. In this way noise can be efficiently filtered.

3.3 Hierarchial algorithms

Hierarchial based clustering methods create a tree structure of the objects in space, where each object is represented as a node, and then either merge or divide these as a means to find clusters. Two competing methods exist with regards to how construct the final representation of the tree structure. The first divides the whole space of objects into smaller pieces, and then merges nodes if a similarity measure has been fulfilled. The second does the opposite,

first constructs a large object which it then divides. Both methods stop when a certain criteria has been met, such as, minimum number of nodes to allow in the tree (the number of final clusters) or minimum number of points to allow for each node (which would then also represent a cluster).

The algorithm known as CHAMELEON does not rely on any fixed statistical model for a given data set when looking for clusters, but instead uses a non-parametric method which adapts to the data at hand [16]. The algorithm operates by constructing graph representation of the data set, where nodes represent data points and weighted edges between these nodes represent similarities. This enables the algorithm to work with both continuous and discrete data sets. The initial step of the algorithm constructs a sparse k-nearest neighbour graph of the data set, while the second step merges these sparse graphs into larger clusters, using two measurements.

The algorithm merges two clusters if they are both relatively nearby as well as exhibit the same shape. The proximity is measured by *only* looking at the points connected to both clusters, as a sub-graph. An average weight value between the connected points is then computed, and used as a reference to determine if two clusters should be connected or not. One advantage of using points only connected to both clusters by means of sub-graphs, is the ability to handle outliers. The shape of a cluster is determined by the connectivity of the points within a cluster. In other words, the edges, representing similarity between the nodes, must exhibit the same pattern for two clusters to merge. As such, the algorithm can distinguish between two *geometerically* different clusters.

As with most other algorithms, CHAMELEON uses a fixed set of parameters specified by the user. The first one controls the the size of each sub-graph in the initial sparse graph. As most of the algorithm's decision making is done by averaging over the weights, this parameter must be high enough for enough nodes to be present in each sub-graph, otherwise the averaging might get skewed. The second parameter can be used to control how the merging is done. The default setting assigns equal importance to proximity and range when determining if two clusters should be merged or not. There might be cases where one or the other should be favoured and these parameters can be experimentally adjusted according to the data being clustered. In their paper [16], they report no major difference for various settings of these parameters, but only a two-dimensional data set was tested. The best-case time complexity is $\mathrm{O}(n \log n)$, but in higher dimensions this value grows to $\mathrm{O}(n^2)$.

4 Problem Statement, Scope and Limitations

Prior to stating the problem a small set of definitions need to be formalized and basic notation of the problem domain explained.

The data sets considered in this problem domain are sampled from continuous distributions. These distributions are assumed to be fixed and do contain a number of statistical outliers. These outliers are usually the result of measurement errors, but could also be rare natural occurences in the distribution from which the samples were taken from. In either way, they need to be filtered in order to make a sound prediction.

The dimensions to predict could be either linear or cyclic. An example of a cyclic data set is the measurement of angles using degrees. These values range from 0 to 360, where 0 and 360 is the same angle. A linear data set has a start and ending point, while the cyclic case has neither. Examples of both are shown in figure 4.

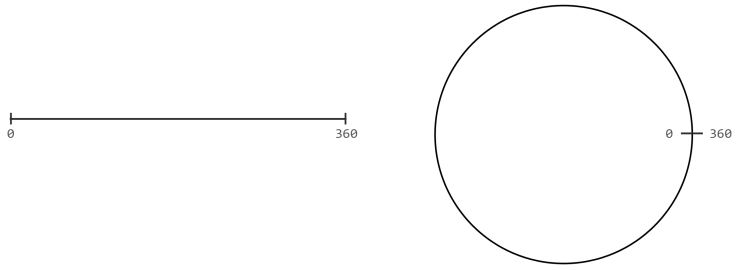

Figure 4: The linear case shown on the left and cyclic on the right.

After these few definitions a problem statement can be formulated as: Given an n-dimensional data set, containing a percentage of statistical outliers, and a dimension y to make the prediction on, what are the likely values (intervals) on y for a list of coordinates $x_1, x_2, x_3, ..., x_{n-1}$ in the data set?

The *main* problem of the thesis is to solve this problem using the algorithms developed by Bergkvist and Damaschke to filter the statistical outliers. As such, instead of finding clusters of data, the method derived will look for anti-clusters.

The *secondary* problem of the thesis deals with finding the most informative dimensions in a multi-dimensional data set. A dimension is said to be non-informative if it shows similar predictions over its range. Both an informative and a non-informative dimension is depicted in figure 5. By reducing the number of dimensions the overall complexity of the problem is also reduced. As such, an effective method is needed to detect the informative

dimensions given that noise is present in the data set. This secondary problem is heavily dependent on finding a robust solution to the main problem of the thesis.

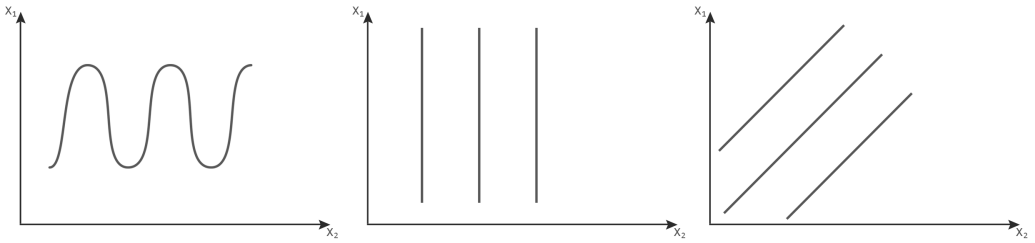

Figure 5: Three figures displaying abstract patterns of data. The left, and right figures show differing patterns depending on the value on the y-axis, and are thus classified as informative. The middle image, however, does not show any variation on the x-axis irrelevant of the value on the y-axis, and is classified as non-informative.

The current *state-of-the-art* clustering algorithms do not provide a satisfactory solution given data sets prevalent in the life-sciences. There are mainly four criteria that must be fulfilled for clustering to be successful in this area, and they are as follows.

Statistical outliers need to be filtered efficiently. The class of algorithms based on k-means, and variants, do not provide a solution at all to the problem of outliers. Both density, and hierarchial, based algorithms, such as DBSCAN and CHAMELEON, can effectively remove outliers, but this is based on the notion that the user provides the correct parameter settings, which can be troublesome in higher dimensions.

Varying density in regions should not pose a problem when detecting clusters. This is one of the difficulties for density based algorithms, as the parameters defining the notion of a cluster is global throughout the data set. The hierarchial based algorithm CHAMELEON can detect clusters of varying size, but again based on a set of parameters which must be specified by the user. In higher dimensions the proximity parameters of CHAMELEON can make this a daunting task.

The patterns in life sciences are not as clearly defined as the artificial sets presented in literature. Usually these data sets contain *simple* geometerical shapes, such as circles and boxes, with noise added using a Gaussian filter. Not only are the patterns simple, they are usually *clearly* disjoint, something which is not common in data sets sampled from the life sciences.

In many cases the user is interested in a small, specified, fraction (a

moment) of the data set. This is connected to the smoothness of the distributions in life sciences, as opposed to the disjoint, more crude, artificial patterns described earlier. As such, there is a need for a clustering method where only a local neighbourhood is being observed, while in the process also be able to filter noise effectively. None of the clustering algorithms presented earlier uses this methodology.

5 Cluster Analysis Framework using DiMaxL

5.1 Rationale

The idea behind the framework developed is to reduce the problem of finding almost empty regions in multi-dimensional spaces, into the less complex problem of finding almost empty regions in one-dimension. This can be done by doing an instance-based clustering method where the neighbourhood around a list of selected coordinates is investigated for similiarities. To help clarify this dense statement an example will be illustrated using only two dimensions.

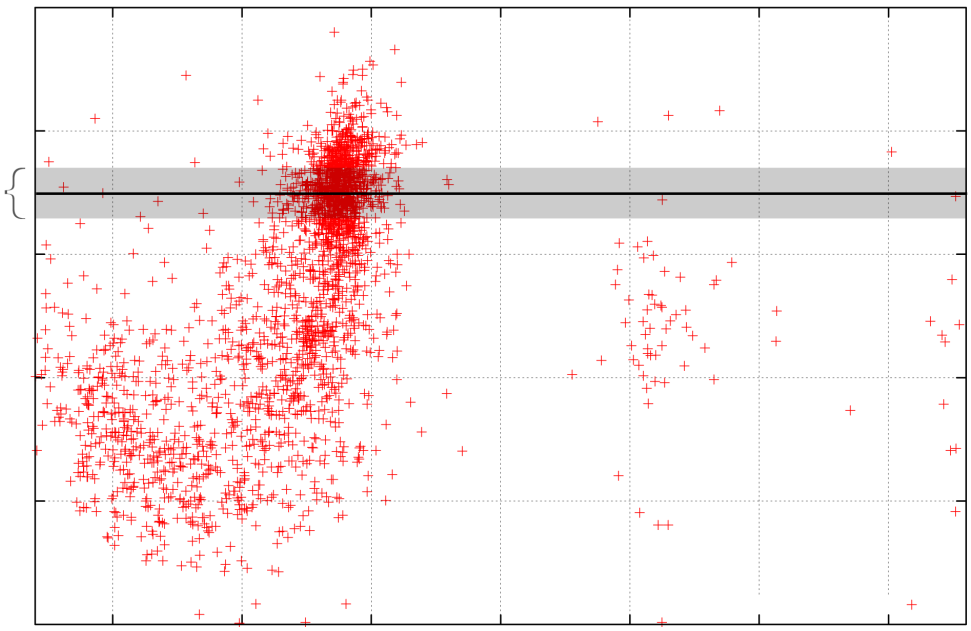

Figure 6: Two dimensions from the Alanine data set. The rectangular grey area shows a selected window.

Figure 6 shows a scatterplot of chemical shifts, taken from a nucleus, versus the torsion angle at which this atom is connected to a molecular structure. The idea is thus to try and infer from the chemical shift values at which position the atom can be connected to this structure. As the values are sampled from continuous distributions, a width around the measured chemical shift, known as a window, will need to be specified. A window can be described as a rectangular region in a two-dimensional space, as can be seen in figure 6. In this case a coordinate (chemical shift) on dimension x_1 has been selected and a width around this coordinate defines one side of

the rectangle. The other side of the rectangle covers the whole domain of the dimension (x_2) to predict. The multi-dimensional case is similar to the previous example, although the points to include is best described by a Venn diagram, illustrated in figure 7. Each circle represents the points included for a single dimension (as described above), and the intersection of these are the points included in the multi-dimensional case.

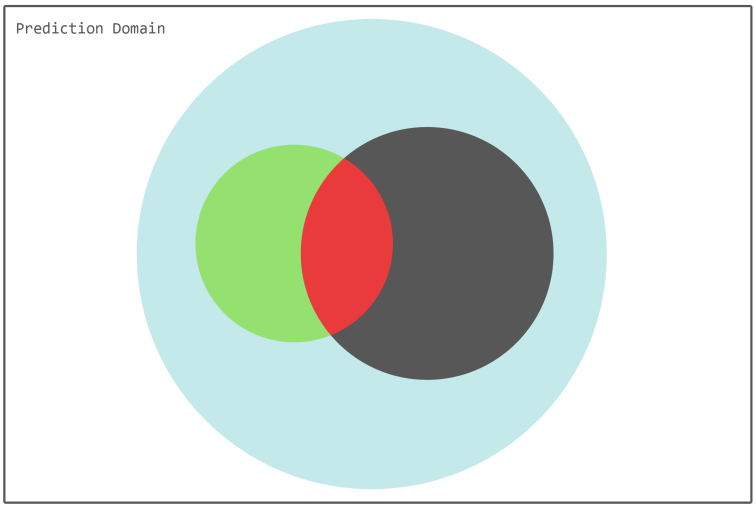

Figure 7: The circles represents valid measurements in the prediction domain and the intersection of these are the considered points in the multidimensional case.

Specifying a too narrow window will result in having too few points to make an accurate prediction (under-fitting). The true local pattern around the specified window may not yet be evident given this narrow width. On the other hand, although specifying a too wide window will yield a large enough number of points to make an accurate (in the statistical sense) prediction, the result will not represent the true local pattern, but instead a more global one. One would thus like to have a window wide enough to include enough points (to avoid under-fitting), but also sufficiently narrow to be able to detect local patterns in the data.

In the optimal case when selecting the width of the window, the sampled data is drawn from either a uniform distribution, or are evenly distributed despite being drawn from a non-uniform distribution. This is, however, rarely the setting and makes the choice of window width complex. A narrow window will successfully detect local patterns in the dense regions of the distribution, while the less dense will require a wider window to yield similar results. As

such, it seems desirable to have a variable choice of window width depending on the density of the region.

There is a trade-off between increasing the width and the accuracy of the prediction. As the width increases more points are added to make the prediction more sound, but the accuracy of the prediction is lowered as a result of this. Once the window is wide enough to overcome the problem of under-fitting there is little motivation to increase the size of the window if the resulting predictions are the same. This is under the assumption that widening the window also adds *more than* just a few points. Once these properties for a selected width have been reached, it is to be defined as *minimal* and *stable*.

Once a window width has been selected we simplify the problem to one dimension. The rationale behind this simplification is due to our assumption that the points in the selected window all belong to one or more local rectangular patterns we would like to detect. To further explain this: assume we have a two dimensional data set, as can be seen in figure 9. A coordinate (and window) has been selected on dimension x_1, and one would like to predict the possible values on dimension x_2 given this selection. As can be seen in the figure, the window exhibits a monotone behaviour on the x_1 axis. As such, finding the rectangle containing the points in this window is equal to finding the interval containing the same points on a line.

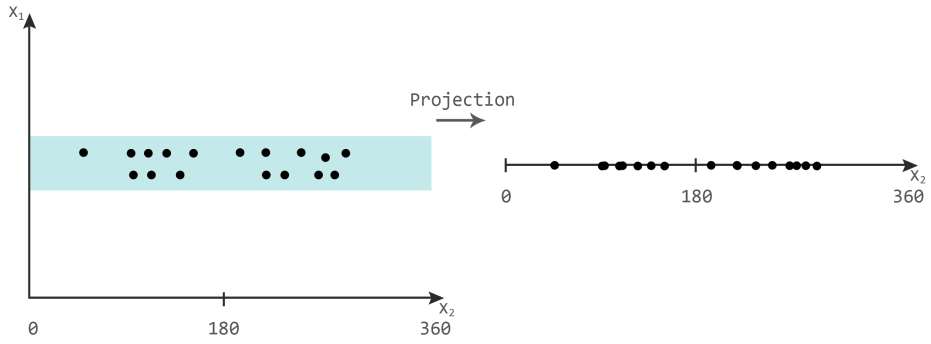

Figure 8: A window being projected.

The most important reason why one would like to simplify the problem to one dimension is that of computational complexity. Finding intervals in one dimension is inherently easier than finding hyper-cubes in multiple dimensions. Once the simplification has been made to one dimension, two different methods can be used to find the intervals containing the data points. The first method focuses on finding the dense regions of data (clusters), while the second method tries to find sparse regions of data (anti-clusters). The latter

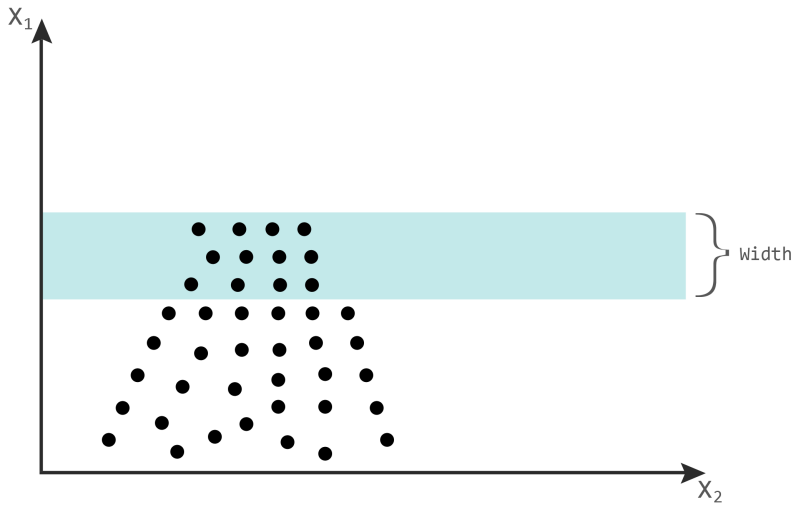

Figure 9: A window has been selected where the prediction on the x_2 axis shows similar results irrelevant of the choice on the x_1 axis.

approach is less computationally expensive and is thus preferred. Despite this advantage both methods share the problem of dealing with statistical outliers.

One way of doing this filtering is by finding the largest empty intervals but with the exception that they may contain a few of these outliers. This problem can been solved in polynomial time by the DiMaxL algorithms, developed by Bergqvist and Damaschke [2], which will be used extensively in solving the problems presented here. A more detailed description of the input and output of these algorithms can be found in the section 5.5.

Given this motivation behind the method being developed, a set of problems have arisen that need to be solved. How does one select the neighbourhood (width) around the coordinates? A measure of similarity between these neighbourhoods, that captures the monotone effects described above, will need to be defined and learned. And last, the number of intervals (empty regions) to look for need to detected.

5.2 Overview

The clustering algorithm presented in this section is divided into four major parts. This overview is meant to give a broad explanation of what the algorithm is meant to achieve in each part as well as introduce some needed notation. As the core of the algorithm uses the DiMaxL algorithms the input

and output of these will be explained as well. Figure 10 shows a flow chart of how the algorithm operates and as can be seen the algorithm works in an iterative way.

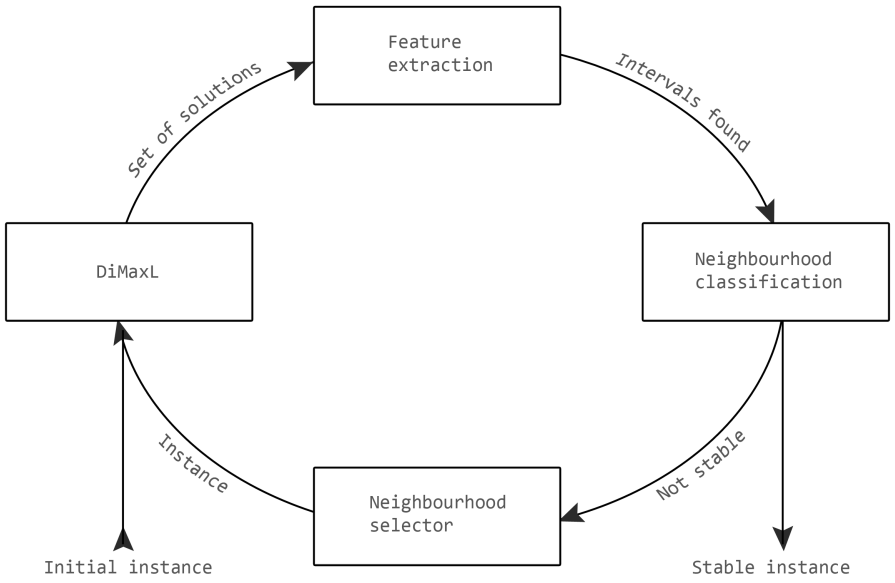

Figure 10: An overview of how the clustering algorithm of the CAFUD framework operates.

During the initial stage of the algorithm, the user has to fix a number of parameters which will both help shorten the algorithmic complexity of the algorithm, as well as make certain assumptions about the nature of the underlying data set.

The first parameter is the *maximum* number of empty intervals to look for, denoted by s. This number does not necessarily mean that the algorithm will always find s number of intervals to be the suitable choice for each instance, but is more a guideline, and will help shorten the run-time of the algorithm. The second parameter is the percentual number of statistical outliers that the user would like to filter. A way to determine the neighbourhood around the coordinates also need to be specified, and this procedure needs a factor by which to stretch the dimensions, denoted by k. The final parameter is the minimum length of an interval, which will be used to remove *artificial* intervals. A more in-depth discussion about these parameters is explained in sections below.

The second part of the algorithm constructs a DiMaxL instance from the initial set of parameters input by the user. The solutions from this instance

is used as input to a feature extraction part, which sole aim is to determine the number of empty intervals to look for.

Once the number of empty intervals has been determined, a classification process is envoked. This process determines if a stable width around the coordinates has been found, and either returns a stable width or the width around the coordinates is increased and the algorithm restarts.

The section detailing how the neighbourhood around the coordinates is selected will be explained prior to the initial phase of the algorithm. This is due to the fact that one of the parameters specified by the user directly affects how the neighbourhood is defined, and thus need to be clarified prior to the initial phase.

5.3 Selecting a neighbourhood around the coordinates

Figure 11: The input and output of the neighbourhood selector.

As detailed in the section 5.1 a width around the coordinates need to be specified. How much should the width be increased at every iteration when looking for this *minimal stable* width? The approach taken to this problem is to, in advance, decide how much the width should be increased at every iteration. The first step in this process is to determine the *lower* and *upper* bound each dimension can *reasonably* assume by removing the statistical outliers at the edges of the distribution. This can either be done manually or automatically. Once these two values have been determined the *span* of the dimension can be calculated:

$$span = upper\ bound - lower\ bound \tag{2}$$

This span would then be divided by some factor $k \in \mathbb{N}$, specified by the user, which would give us the *delta* width at each iteration (from now on this value is denoted by δ).

$$\delta = \frac{span}{k} \tag{3}$$

By doing this for each dimension we ensure that they are all stretched uniformly according to their distribution. Although the method is crude, it is simple, and allows the user to control an important aspect of the results, namely the minimum accuracy by which a prediction can possible have. This parameter obviously also determines the level of granularity by which the algorithm can detect patterns. Setting this value too low and the finer details might be missed.

5.4 The initial phase and user parameters

Most of the parameters given by the user can be automatically infered by running a number of tests on each instance, but the algorithmic complexity of DiMaxL would require a much longer running time (per instance) compared to simply allowing the user to specify these manually.

The first parameter specified by the user is the number of empty intervals, s, the algorithm should maximally look for. This parameter could be automatically infered if the algorithm would be run $O(n)$ number of times for each instance, under the assumption that the feature extraction part is able to determine the number of intervals for each instance. Some knowledge of the data set is thus required by the user, as this parameter is specified manually.

Since the data sets do contain a number of statistical outliers, the algorithm needs to know how much of the data is to be considered as such. A number of ways can be used to specify the amount of outliers present in the data set. The reasonable measurement to use is a percentage based value, given the amount of points currently being considered. To clarify this part an example will be used: Assume a data set contains 1000 valid data points, and 2% of these are considered as statistical outliers. A neighbourhood is selected where 200 of these points are included, and as such 4 of these 200 points would be considered as outliers. This value is thus relative to the local neighbourhood.

Previous section outlined the purpose of the parameter k, known as the factor.

The last parameter deals with smaller intervals, which may be considered artificial holes, possibly occuring due to unexpected regularities in the sampled data. These artifical holes are present in areas (on the line) where there is otherwise a clear pattern (either present or emerging). The feature extraction tool, which deals with finding the number of correct empty intervals to look for, may have problems with these unexpected regularities, and thus needs a parameter that specifies the minimum length for an empty interval. The rationale is that the user, who in advance already knows parameter s, will also have an idea about the length of the intervals, and can thus help remove some of the shorter ones which may occur as described above.

Once these parameters have been specified, and a list of windows selected, as well as a dimension to predict, an initial DiMaxL instance is created.

5.5 Solving an instance using DiMaxL

Figure 12: The input and output of the DiMaxL algorithms.

The first task is to find all the points to be included in the given instance, as specified by the choice of windows, and project these onto the dimension one would like to predict. This new one-dimensional problem of finding the almost empty regions is now solved using DiMaxL.

Although the user specifies the statistical outliers in percentage, the algorithms work with whole integers when specifying the number of points to regard as such. Bergkvist and Damaschke denote the number of points to regard as outliers by p, and the same notation will be used here. Below is a set of notation connected with the DiMaxL, which will be used throughout the remainder of the text.

$$s = \textit{number of intervals to look for}$$

$$p = \textit{the fixed number of points to regard as outliers in a given instance}$$

$$n = \textit{total number of window points in an instance}$$

$$p/n = \textit{percentage based number of points to allow in an instance}$$

The algorithms use a dynamic programming scheme and thus return the solution for every combination of $i = 1, .., s$ (intervals) and $j = 0, .., p$ (outliers). For every solution the length of the intervals, the number of points included, as well as the starting and end point of these are returned. These sets of solutions will be used both by the feature extraction tool, to figure out the correct number of empty intervals, as well as by classifier when determining the stable width.

5.6 Extracting the correct number of intervals

Figure 13: The input and output of the feature extraction tool.

As the method developed is motivated by comparing the almost empty regions, and not the dense ones, care must be taken when selecting these. In regions where density is high, adding an superflous interval will do nothing more than perhaps increase the total length of a solution by a small amount. In less dense regions, however, adding an interval may increase the length of the total solution by a greater margin. Although one may not see this as a huge issue when running a single instance under the supervision of a user, this problem creates obstacles when the framework is to find the informative dimensions.

The problem is related to missing and (or) lack of valid values in higher dimensions, resulting in high fragmentation. The same problem can be observed, in lower dimensions, if the factor used to stretch the dimensions is too high, which results in few points being added each iteration. In situations where this occurs the minimum length of an interval plays a crucial role to try and minimize this effect, but is not a complete solution to the problem. In a certain sense, this problem stems from the fact that the algorithm is looking for anti-clusters, which will be less affected if fewer points are added each iteration, as the outliers will be even less common in this case.

There are a few assumptions which can help detect the correct number of intervals. The first assumption was touched upon previously, and is that of fragmentation. If an instance on the line is highly fragmented, the solution's start and end point can reasonably be assumed to shift a lot. Put differently, the intervals will tend to jump around, for different values of p.

At first an attempt was made to try and create an intermediary solution, for a given instance, which would capture this behaviour. The idea was to create, from the set of solutions returned by DiMaxL, a more *monotone* solution, using a set of rules and assumptions. This monotone solution would try and fix the intervals at a given position (to the best of its ability), and those intervals which, despite these rules, did still jump around, would be regarded as emerging patterns and could be discarded as empty intervals. The combinatorial nature of the problem itself did, however, result in many

heuristics for the many special cases. Although this method was successful in lower (2-3) dimensions, where fragmentation was relatively low, it did not work well in higher dimensions. During this phase the *artificial* holes were discovered and the choice was made to allow the user to specify a minimum length for the intervals.

Instead of recreating a monotone solution, backtracking the set of solutions can give us an insight into the number of likely intervals for an instance. The idea is to find the solutions, in the set of solutions, which can give us hints as to how many intervals there should be. The algorithm is recursive and can be explained as follows: Look at the solution with s=s and p=p. Inspect each interval and if one of them is less or equal to the length of an *artificial* interval, as specified by the user, then we reduce the number of intervals by one, and we start over again. If none of the intervals can be considered artificial then we compare them for adjacency instead. Given that two intervals are separated by a single point (they are connected to the same point), then we asssume one interval to be artificial and reduce the number of intervals by one, and start over again. If none of the above applies, then we try and reduce the number of points instead. This will, however, only be done if the resulting solution (with one less point) has similar length as the previous one. The whole algorithm is shown below in pseudo code.

```
intermediarySolution( int : s, int : p ) : void
if s > 1 then
  for all intervals as i do
    if i is considered artificial then
      return intermediarySolution(s − −, p ← initialPoints())
    end if
  end for
end if
if p > 0 then
  if solution with one less point is has similar length then
    return intermediarySolution( s, p– )
  end if
end if
if s == 1 and p == 0 then
  p ← min(3, initialPoints())
end if
endintermediarySolution
```

Once the algorithm has finished the assumed number of intervals for a given instance is returned.

5.7 Defining and classifying neighbourhood similarity

A measure of similarity between instances needs to be defined, and learned, in order to find the stable width. As previously noted, the number of intervals for a given instance plays an important role with regards to the length of solutions, and it is partially the length of solutions which will be used to create the measurement of similarity between the instances.

Prior to stating the measurement in detail some brief notes will be made as to how this measurement was conceived.

In the most extreme case a width of zero is selected around coordinates chosen by the user. Most likely, given the fact that the measurements are taken from continous distributions, very few (if any) points will actually be contained within this narrow window. By gradually increasing the width, the pattern around the coordinates will start to take shape. If the DiMaxL algorithms would be run on these instances, the reasonable thing to assume would be to see very different solutions until the *true* pattern is starting to clearly show. One way of determining if a stable width has been found would thus be to try and classify when this true pattern has finally emerged.

The first measurement used was the correlation between number of points included in the intervals versus the total length of solutions. To be more precise, the number of points included was specified as the ratio of $\frac{p}{n}$, and the total length was in percentage, to normalize the learning procedure. A graph showing this correlation for two instances is displayed in figure 14. The assumption was that these two graphs would have similar shape once the true pattern emerged, while they would not have similar shape prior to this.

A way to measure similarity in this case was that of average deviation. If two graphs look similar the deviation between points on the graph will remain roughly the same throughout the graphs. The graphs which would not look similar to one another would deviate from this average deviation, and add to an error value. This error value would in turn be used to classify two given instances as either stable or not stable. The error function used to capture this behaviour is the *mean squared error* function, which is shown below.

$$E = \frac{1}{k}\sum_{i=1}^{k}(\hat{d} - d_i)^2 \tag{4}$$

The value k is the number of points used to compute both the average deviation, as well as the total error. The average distance is denoted by $\hat{d}$, while the distance at point i is denoted by d_i.

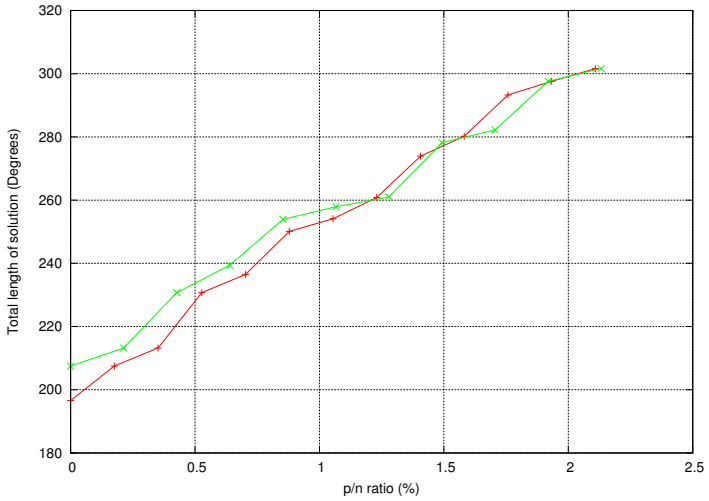

Figure 14: A graph showing two instances compared using their p/n ratio versus the total length of solutions.

At first, a k-nearest neighbour (KNN) classifier was used to classify if two instances would be considered stable or not. Although the KNN method can be an effective classifier, if the different classes are clearly defined, it has difficulty capturing more advanced patterns of classification. This became prevalent in situations where few points were included in the first few windows considered, which would often take place when multiple dimensions where used. To remedy this the δ would have to be increased by lowering the factor. This in turn resulted in a *much* less accurate prediction that did not represent the true pattern around the selected coordinates.

Instead of using a KNN classifier, an artificial neural network (ANN) was used to try and learn the more complex nature of the error function. Although the error rate, when testing the classifiers, was lower (close to 98% correctness) compared to that of the KNN classifier, empirical tests still showed the same flaws as described above. The hypothesis of how to accurately describe when two nearby instances are stable had to be revised.

A few important factors made the previous assumption about how to describe stability between instances flawed, and these were noticed during the testing of the KNN and ANN classifiers. First off, the graphs did not show enough variation (with regard to deviation) to be able to use the average deviation as a measure of stability. In many cases, as described above, too few points could be used between instances to make an accurate classification. Secondly, the measurement itself did not correlate to the width, nor

the amount of points, but instead to the behaviour of the graphs, which is unpredictable given the nature of the problem. In many instances where one may have classified two instances as stable, the graphs would not fully correlate. These instances had one thing in common though, they would usually end up at the same total length (given maximum value of p), while the first moments of the graph would be *chaotic*. This is to be expected, however, and gives a hint at what to really measure instead.

The commonality between two neighbourhoods (widths) is better measured by the average distance between them. In other words, a small distance between the graphs of two neighbourhoods is a good measure of commonality. The average distance error function, as described earlier, is only valid if enough points (p) acts upon the error function, but this becomes a catch 22, given the small amount of points during the initial widths. And as one increases the width, the classification process becomes a tautology, due to the facts described above, where it only really relies on the amount of points in the window. The new error function thus look as follows.

$$E = \frac{1}{k} \sum_{i=1}^{k} d_i^2 \tag{5}$$

There is no average distance at all, but instead only a distance which is squared to account for negative distances. As with the previous classifer, an artificial neural network was used to learn this error function. A feed-forward back-propagation network was used, using three hidden layers containing three neurons each.

5.8 Description and pseudo code of the algorithm

Using what was previously described, with regards to selecting the number of intervals and classifying widths, the full algorithm can be presented and described.

As input the algorithm takes the number of intervals to look for, the percentage of outliers (pn), the list of coordinates, a factor to stretch the dimensions with (k), and the instance which is returned (and thus needs to be a pointer or reference). The algorithm returns the number of intervals for the computed instance, or zero if no valid instance can be found.

Instead of describing every instruction in the algorithm, a broader description of the events are presented instead. The actual creation of the DiMaxL instance assumes that the algorithm *knows* about the cluster in question, but could obviously also have been used as an input parameter.

The algorithm starts by creating the initial instance. It then checks to see if we have enough points in the window currently being looked at, and returns if we don't have enough points. This is a check which only matters during the first iteration of the algorithm, but given that it is not a major performance issue, it remains each loop.

When the first few checks are done the algorithm solves the instance using DiMaxL, computes the number of intervals, and adds this to a list of instances (*adjacent*). If this list contains enough (two) instances some further checks are made, prior to classification. If the two instances contain the same number of points p, then we increase the width a bit, as these instances can be considered artificial and will be classified as stable no matter what.

Once this is done we check fragmentation. This is done by comparing the solutions containing the maximum number of points (p) from each instance. If these intervals do not intersect with one another, they are considered too fragmented and we increase the width.

Finally we check to see if the instances are stable, by using the classifer as described above.

```
subseq( int : s, double : pn, [Coords] : coords, int : k, Instance : ins ) :
int
i ← 1
[Instance] : adjacent
while i ≤ k/2 do
  Instance : currIns ← new instance using s, pn, i, coords
  if too few points in current window then
    return 0
  end if
  solve currIns using DiMaxL
  compute and set number of intervals for currIns using intermediarySo-
  lution
  append currIns to the adjacent
  if size of adjacent = 2 then
    if both instances have equal number of points p then
      remove oldest instance from adjacent
      i ← i + 1
      continue
    end if
    if the intervals from the last solution from both instances do not
    match then
      remove oldest instance from adjacent
      i ← i + 1
      continue
    end if
    if the instances are classified as stable then
      instance ← currIns
      return number of intervals computed for currIns
    end if
    remove oldest instance from adjacent
  end if
  i ← i + 1
end while
return 0
endsubseq
```

5.9 Finding the informative dimensions

A dimension is considered non-informative if the pattern remains the same over its whole range. As such, it can be removed from a larger set of dimensions without altering the outcome of the pattern. The clustering algorithm presented in the previous section will be used to detect these non-informative dimensions.

One popular method used, when reducing the number of dimensions, is that of stretching the dimensions [19].

The idea is to make two queries, where a query is defined as an instance to be solved by DiMaxL using the previous algorithm, and compare these. One query containing the full set of dimensions and the other with one of the dimensions removed. The results of these two queries are then compared, and if they are equal one can conclude that the dimension being considered does not add any relevant information to the query. By querying a large enough sample on the data set, a pattern can start to emerge, which in turn can be used to classify if a dimension is informative or not.

Instead of considering every solution, for every given p, only the one with the highest p will be considered when comparing two queries. The motivation behind this is two-fold. First, it is motivated by how the similarity between instances was defined earlier, which dictates that most solutions will be similar in length. Secondly, the most reasonable solution to select, of all solutions, is the one with the highest p, as the intervals will be maximized in this solution. The hope is that the neighbourhood selected by DiMaxL as stable is so dense, that even maximizing the number of points p will not affect the total length of the solution drastically. This can not be guarenteed if some other solution, with fewer points, is selected.

The error function used when finding the informative dimensions is equal to that of finding the stable width. The distance d_i is the comparison of two instances with one dimension removed. The number of tests used to classify, wether a dimension is informative or not, should be specified by the user as a percentage of the valid instances. This value is denoted by k below. Naturally, the higher the percentage the more accurate the reduction will be.

$$E = \frac{1}{k} \sum_{i=1}^{k} d_i^2 \tag{6}$$

There are a few things to overcome when making the reduction. As the queries used as reference will have to be randomly selected from the data set, care must be taken to select a factor that encompasses enough valid instances. This is a huge issue in higher dimensions, if the data set is not

large enough, or if there are a lot of missing values. As the reduction removes dimensions, theoretically the factor could be increased (thus lowering the δ), but an automatic choice of this value is difficult to establish. To clarify, in higher dimensions a lower value of the factor will usually tend to be needed, thus lowering the accuracy of the prediction. This factor may be so small, that the δ at each iteration is too large to detect the finer details of the patterns, thus resulting in similar predictions (over-fitting) over the whole range.

Two solutions exist to this problem. One either uses the same value throughout the whole reduction, or a different value is specified, once a reduction has been made. The first option is less accurate, while the second one requires intervention from the user at each reduction, thus potentially increasing the already time consuming problem by an arbitrary factor.

If two dimensions are tested and classified as non-informative, as described above, the dimension with the highest error value should be removed and the test re-run. This increases both the run-time complexity factor, as well as the assurance that the correct dimensions are removed. Assume there are five dimensions, excluding the dimension to predict, and we check the data set for informative dimensions by removing each dimension once. The dimensions classified as non-informative would then be ranked, and the dimension with the highest error value would be removed from the data set. The algorithm would then re-run the test with the remaining four dimensions, and so forth.

6 Results

6.1 Stable width classifier

In this section results relating to the stable width and the ability of the framework to accurately detect patterns and remove noise, for various choices of factor, are presented. As the multi-dimensional case is difficult to illustrate, only two-dimensional examples are presented. The multi-dimensional case is, however, not different from the two-dimensional one, given that the measured data is taken *as-is*, and then projected to a single dimension. The problem in higher dimensions using this method, is the lack of valid values, but this can be simulated by means of increasing the factor in the two-dimensional case.

Figure 22 shows the results from the artificial neural network used to classify the stable instances. Close to 600 instances were used in the training set, and out of these a third were used in a cross-validation set.

A selection of various types of patterns found in the data set from the protein measurements are illustrated. These selections are presented using two figures containing two images.

The first represents the data set, illustrated by a scatterplot, while the second figure shows the total length of solutions given a specific coordinate and factor. These results have been generated by taking 20 equidistant points (in relation to its nearest neighbour) over the dimension's whole range. The factor ranges from 10 to 150, where a higher factor gives a lower δ width. By generating the results this way, all types of data (dense or sparse) is ensured to be included. The *artifical holes* parameter was set to 3% of the total prediction length.

The second figure shows how the algorithm interprets the data set as it projects the windows onto a single dimension. Both the pre-processed instances, containing noise, as well as the post-processed, are displayed. These figures have been selected at various factors to illustrate some of the strengths and weaknesses with the method used. The last figure illustrates what happens when the noise ratio is increased.

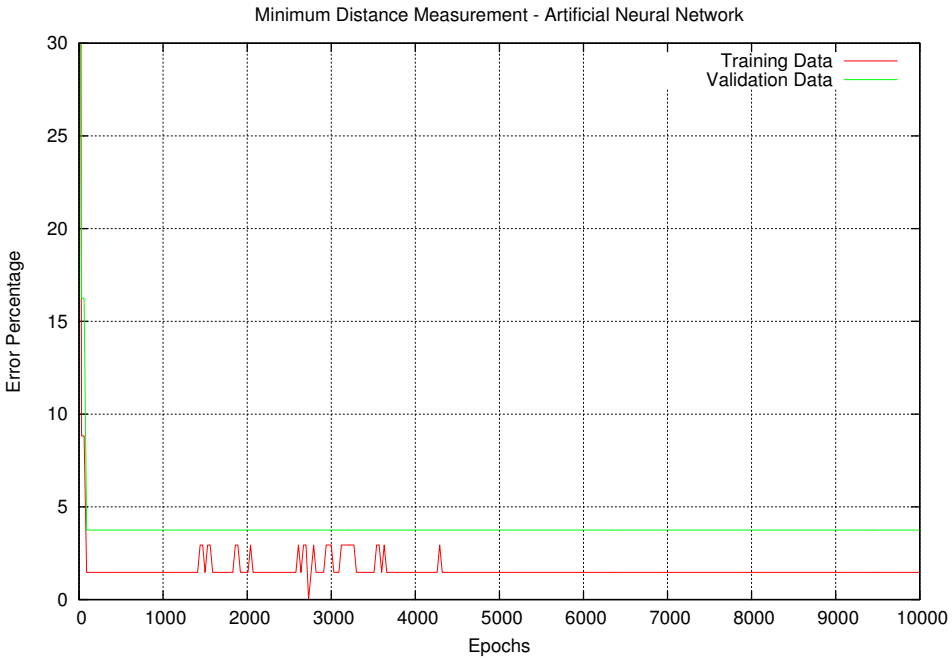

Figure 15: The error rate of the ANN using the training and cross-validation set.

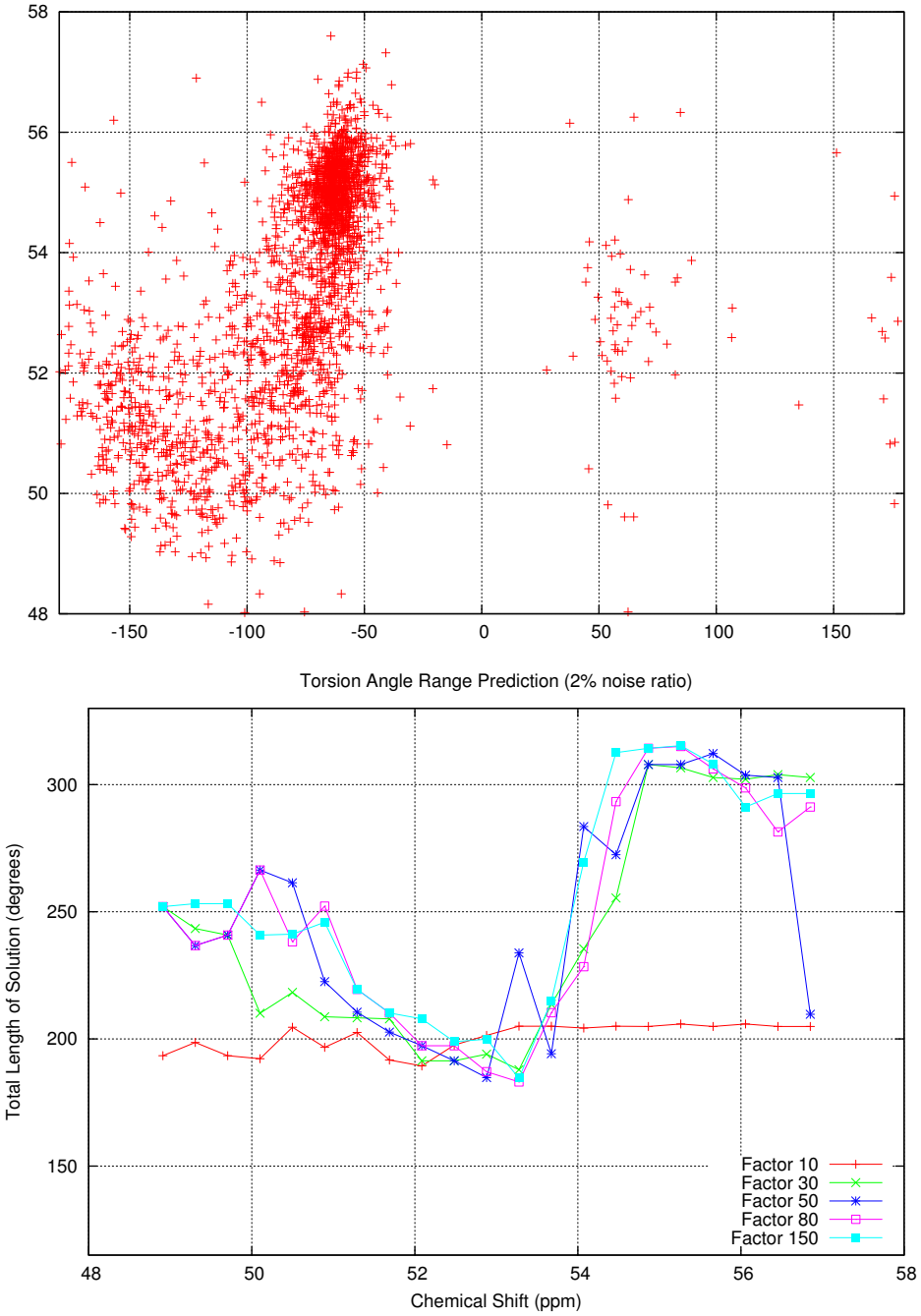

Figure 16: The upper image showing two dimensions from the Alanine data set, while the bottom images displays the length results of solutions for various factors.

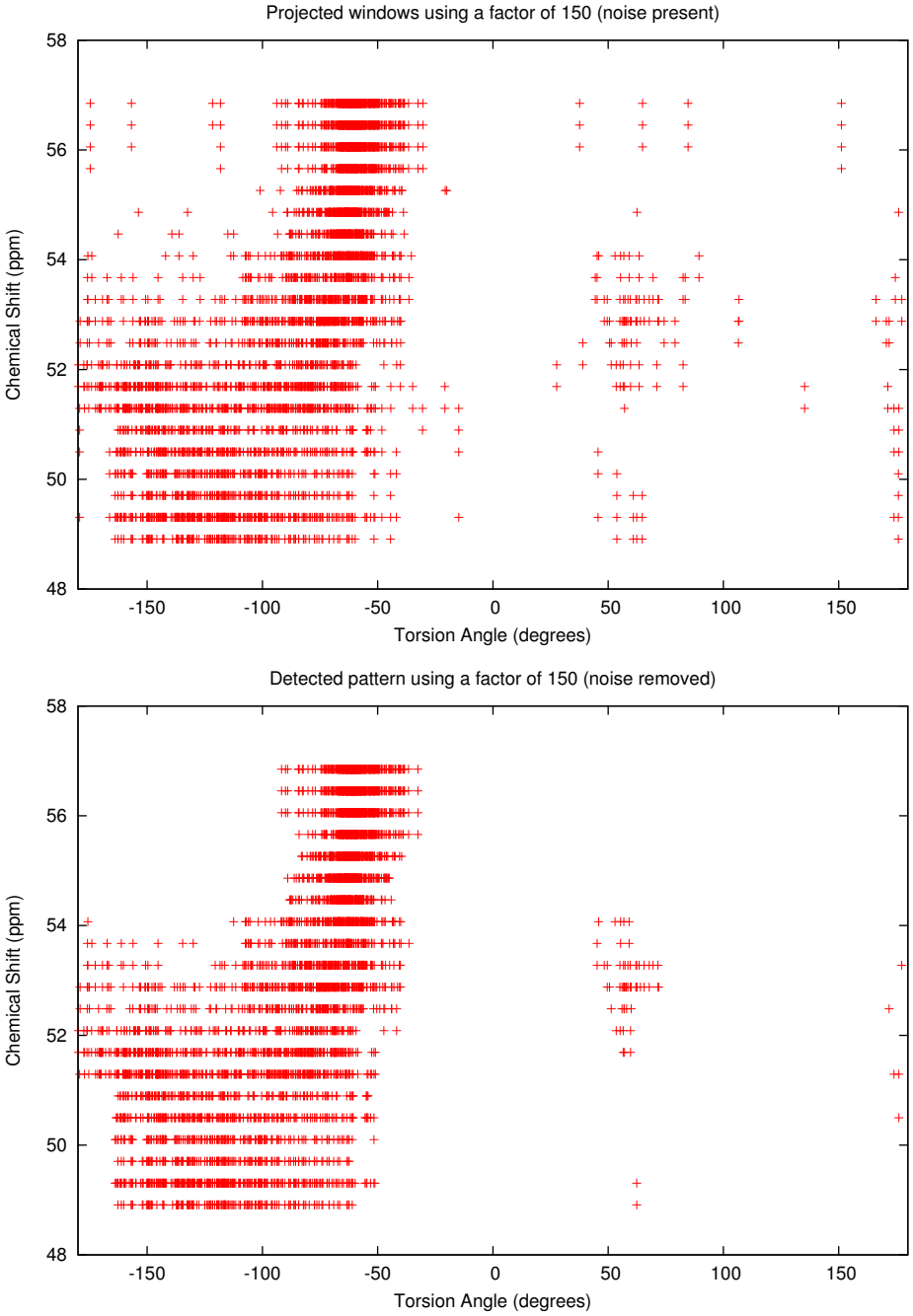

Figure 17: The upper image shows the raw instances, as input to the DiMaxL algorithm, and the bottom image shows noise removed (final result).

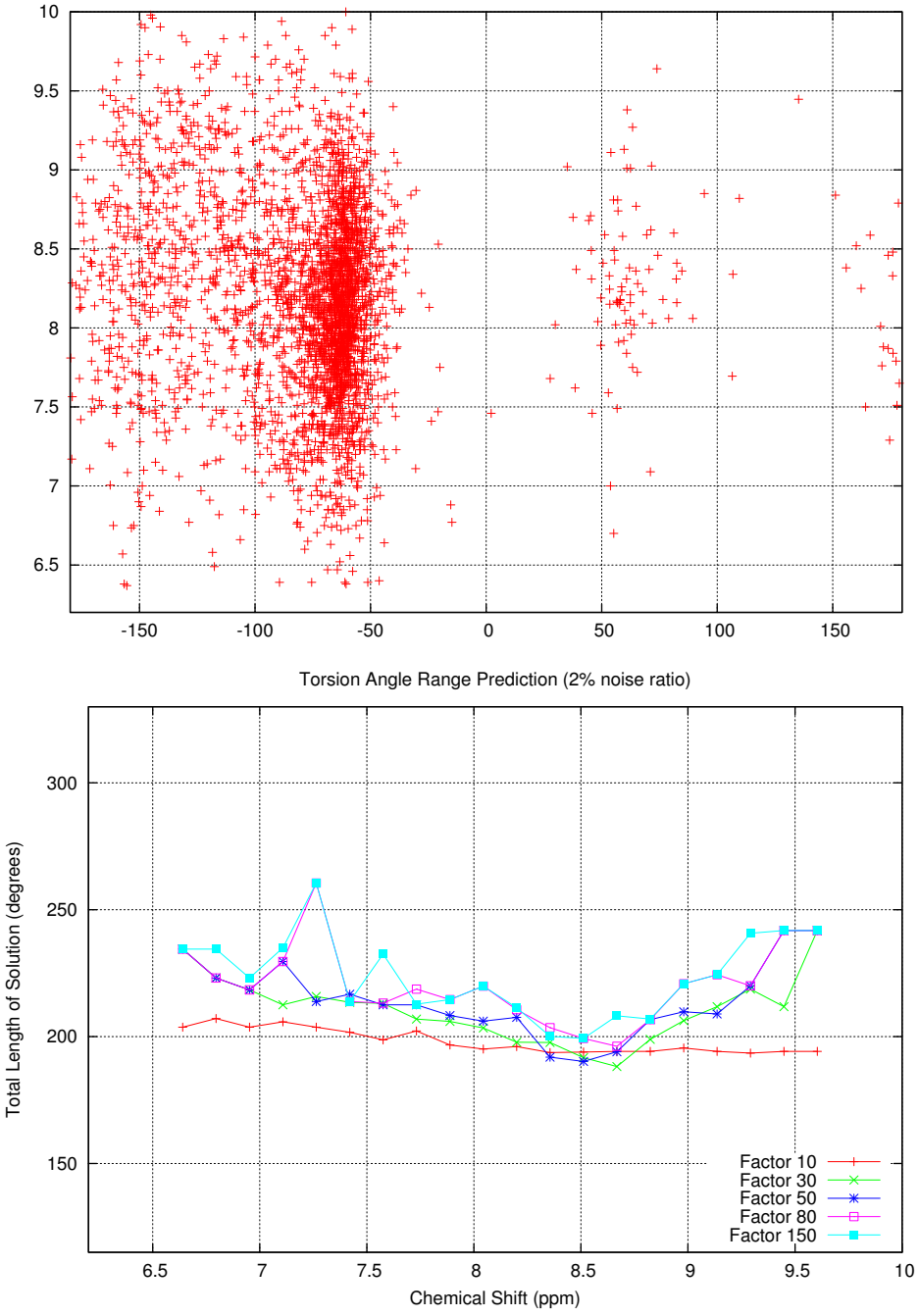

Figure 18: The upper image showing the raw Alanine data set, while bottom images displays the length results for various factors.

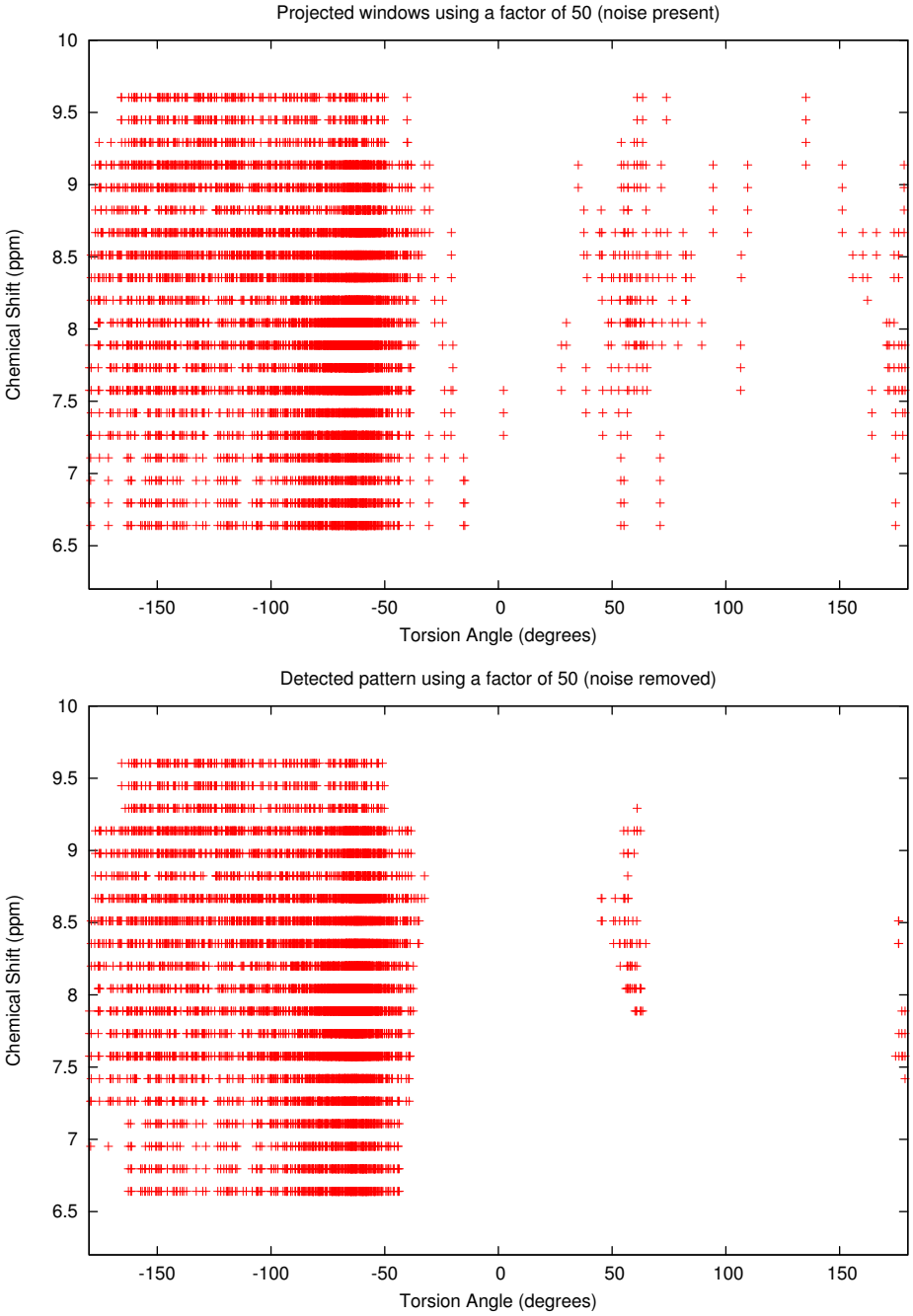

Figure 19: The upper image shows the raw instances, as input to the DiMaxL algorithm, and the bottom image shows noise removed (final result).

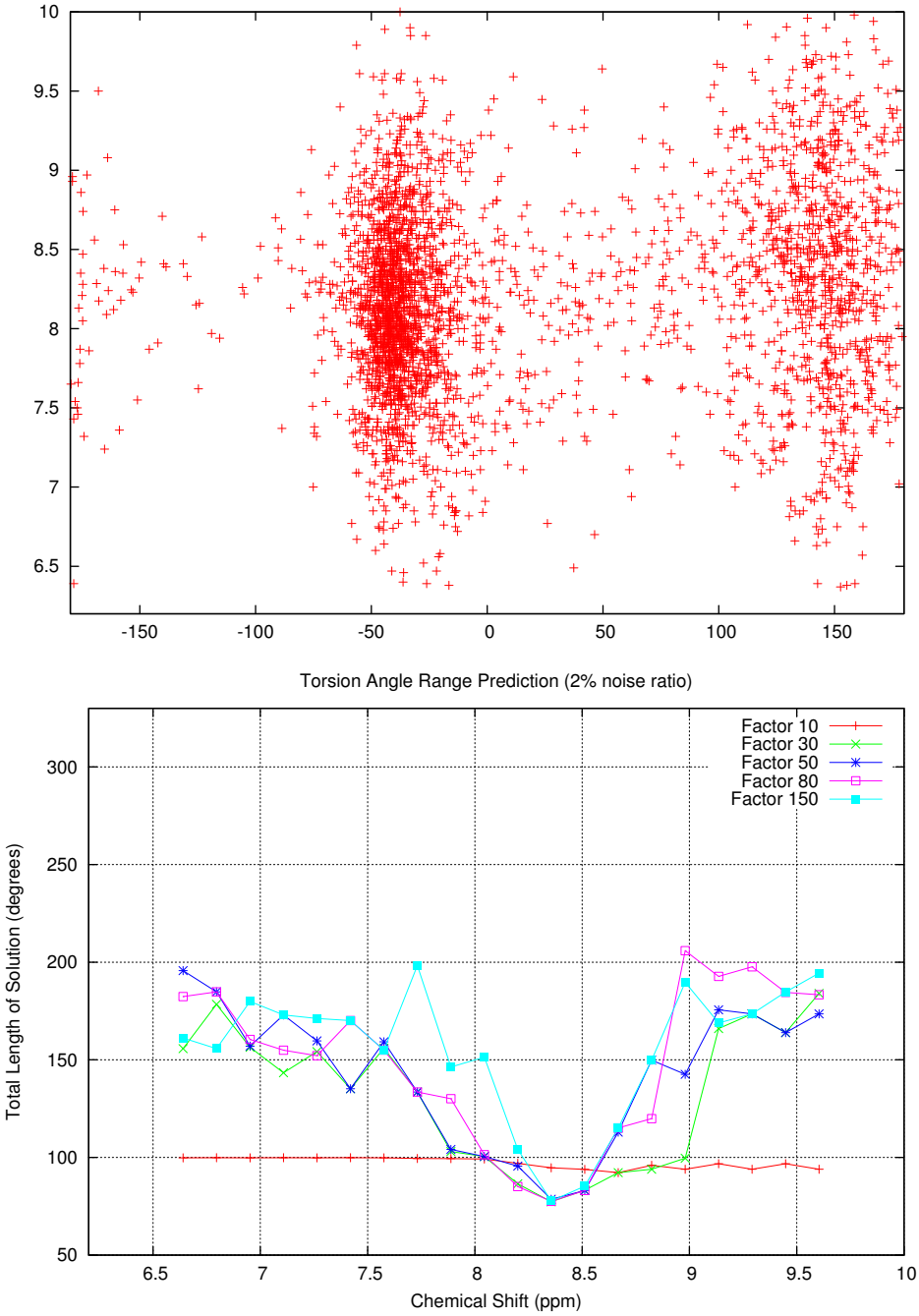

Figure 20: The upper image showing the raw Alanine data set, while bottom images displays the length results for various factors.

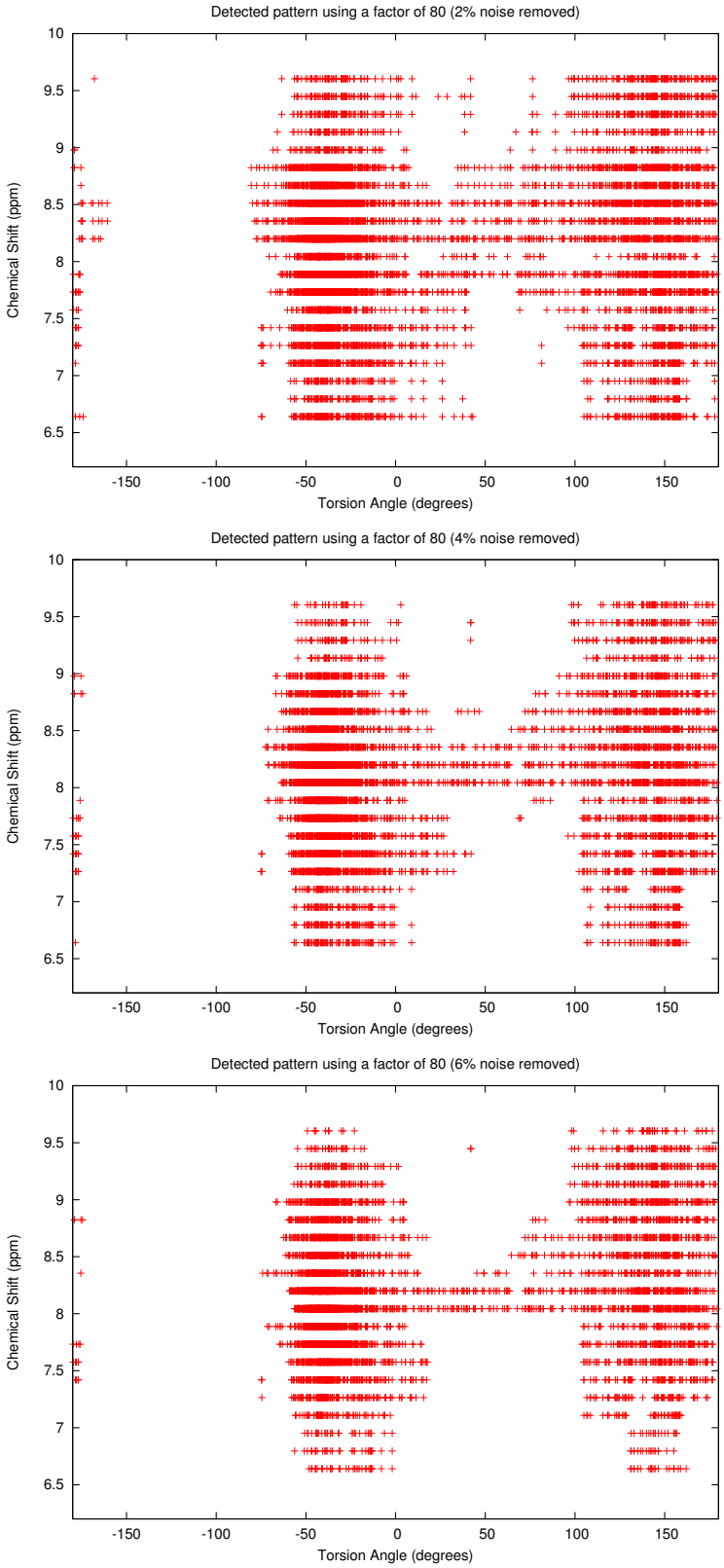

Figure 21: Three patterns emerging with varying selection of noise.

6.2 Informative dimensions

The first figure displays the success of the artificial neural network in determining if a dimension is informative or not. Around 130 instances were used to train the network, and out of these a third were used for cross-validation. A factor of 30 and a noise ratio of 3% was used to generate these results.

The two final figures show the results of the informative dimensions classifier on two dimensions from the Alanine data set. During each iteration one dimension is removed and compared with the full data set. If two graphs deviate from one another, then the dimensions are informative, otherwise non-informative.

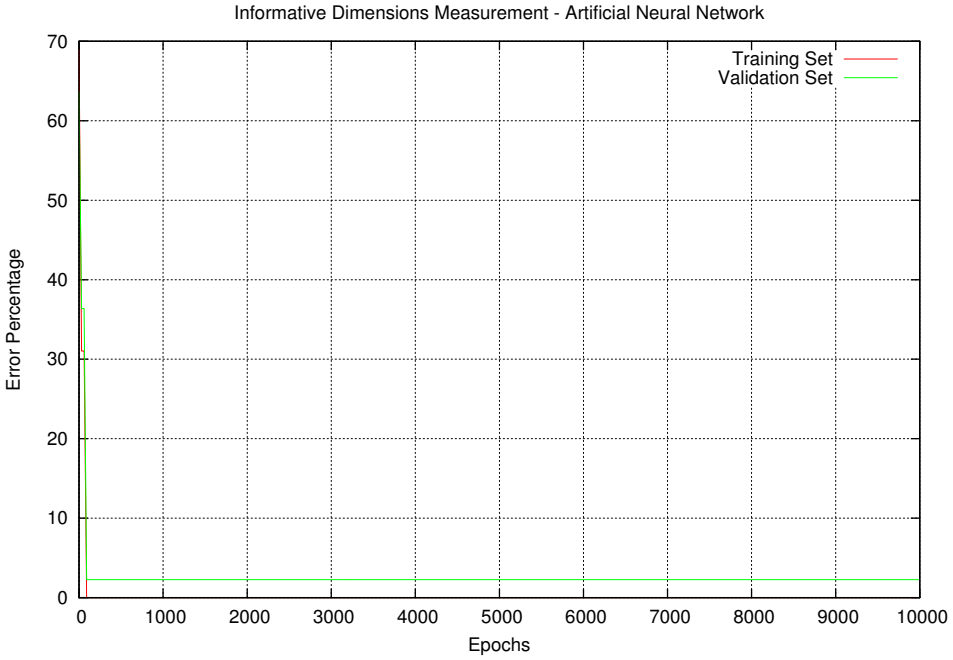

Figure 22: The success rate of the ANN on the training and validation set.

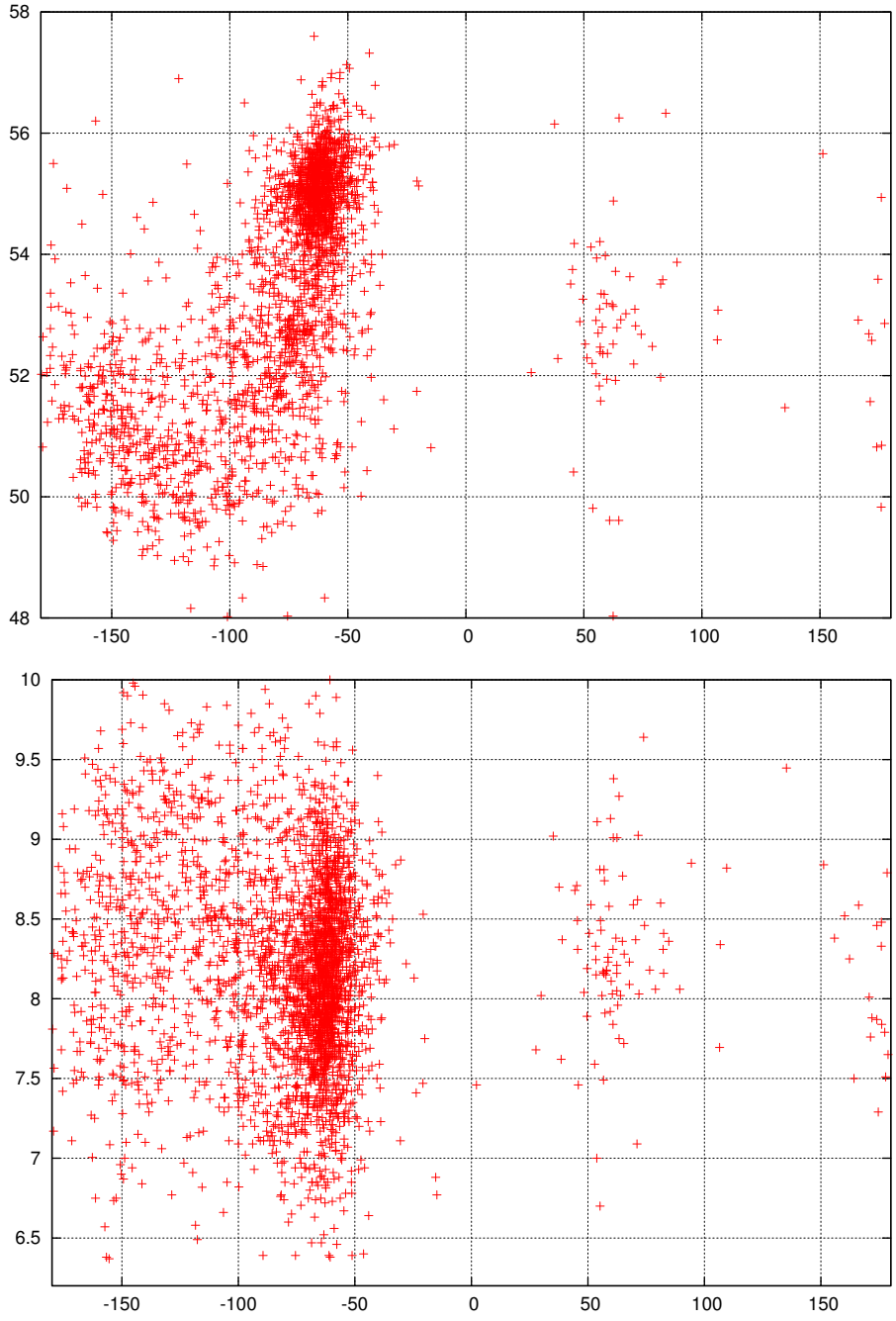

Figure 23: The upper image displaying dimension 0, while the bottom displays dimension 3.

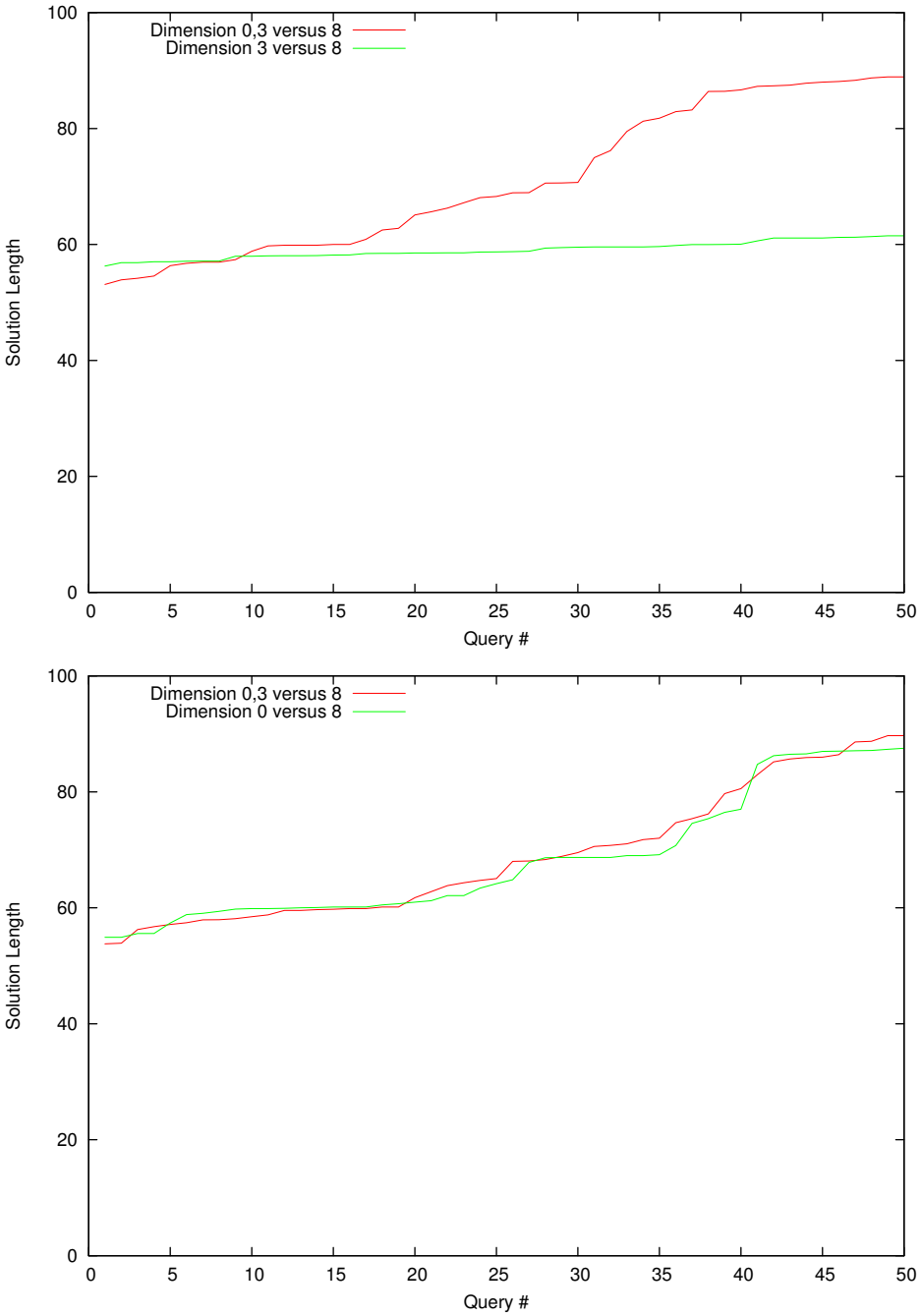

Figure 24: There is little change in the results wether or not the third dimension is added, as can be seen in the bottom image. The upper image shows the opposite, however, and marks dimension zero as informative.

7 Discussion

7.1 Stable width classifier

The ANN used to learn the minimium distance function quickly converges to a low error rate. This is to be expected, however, given the simple nature of the function. Because of the fact that the training examples were classified manually, some faults may have occured during this process, thus making the error rate of the training and validation set hover around 2% and 4%, respectively.

Given how DiMaxL operates, only the truly sparse regions of data is filtered, assuring that the *true* regions of dense data remains intact. Even when increasing the percentage of outliers to filter, the pattern detected by the algorithm is more or less same, as can be seen by the last figures in the results section. In some parts of the these plots, even when as high as 4% noise is removed, there are still sporadic points scattered. Although these may look like individual points, the likely scenario is that there are two tightly connected points, which make it look like one. Either way, these sporadic points do not affect the overall result by a large margin, as the area around them will be classified as empty.

The method also seems to be invariant to differing choices of factor. Although there are cases where the factor does give differing answers, as a whole the results are the same. Only when the factor comes near 10 do the results seem to overfit. When the *average distance* measurement was used as a means to detect the stable width, under-fitting was the most difficult problem to overcome, but this does not seem to be major issue when using the *minimum distance* measurement. The reason for this is two-fold. Not only do we measure stability by means of graph comparison, we also measure it by looking at the location of the empty intervals. This effectively seems to counter the lack of measurable data available when there are too few points in a window.

Most results show a similar pattern compared with the original data, even when more complex patterns are present. There are, however, exceptions to the rule. Figure 21 displays an almost rectangular pattern as a result, while the bottom left area of the original data, between 6.5 and 7.5, could be perceived as noise and not valid data. One reason why this overfitting has occured is the the choice of factor. A higher one would most likely have detected this less dense area as noise, as can be seen by previous patterns detected with a factor of 150.

Although the results do not present a comparison with the current state-of-the-art algorithms, some points can be made with regards to their per-

formance on data sets from the life sciences, compared with the method presented in this paper. The k-means variants will not be dealt with at all during this comparison, given the fact that they do not filter outliers at all.

Both CHAMELEON and DBSCAN (including variations) can effectively remove noise, but fine-tuning their respective parameters can be troublesome in higher dimensions [9, 16]. The DiMaxL algorithms work in a much more crude way, but ensures noise removal where density is low. Setting the percentage of outliers to remove slightly higher than the expected number, does not seem to alter the detected pattern by a large margin either. By only having one parameter controlling the level of noise to remove, the complexity of the final application is lowered.

Most algorithms, except those based on kernel methods, operate on the whole data set, when perhaps only a small fraction is of interest to the user. This creates a large overhead when huge amounts of data needs to be processed, as the algorithms, in worse case, have a complexity of $O(n^2)$ [16]. In the current data sets tested, the algorithm would usually find a solution with less than 1000 points in the window, ensuring fast execution. Although no large study has been made to back up this claim, the large number of tests done during the development of the framework does point in this direction.

It is not the number of parameters required by the framework, which makes it simply to use, but the simplicity of the parameters. Many other frameworks rely on the user to make assumptions about the distributions, and/or detailed knowledge about the sometimes complex parameters that need fine-tuning. This added complexity has its pros and cons. The parameters, although complex, does add flexibility to the frameworks, but at the expense of usability. The framework presented does not have many complex parameters, although some knowledge about the data is required by the user. This makes it, hopefully, not only easy but also an effective method for practioners in areas such as life sciences, where the task is find patterns in data where other methods have previously been less successful.

7.2 Informative Dimension

As with the stable width classifer, there is little problem for the ANN to converge to a low error rate. The training set is lower than for the stable width set, but this can be attributed to the time consuming process of producing a single instance. Each instance is constructed by running around 100 random queries, making the total number of instances executed a bit over 13000.

The results show similar results for dimension three, and this is to be expected, given the symmetric nature of this dimension. Every solution hovers around 60% length, while dimension one varies between 55% and

90%. These results show very clearly a non-informative versus informative dimension, but in some cases this may not be as evident.

This is especially difficult in less dense areas of distributions, and where missing values are common. As such, care has to be taken when selecting the factor when reducing dimensions. The optimal case would be to, at every iteration, select a different factor which is most suitable for the specific reduction. This can be time-consuming, however. Another approach, to remedy the lack of valid instances, would be to use a less strict procedure when determining if an instance is valid or not. At this point, the neighbourhood at the first iteration is used as a reference, but maybe one can increase this starting neighbourhood when determining if an instance is valid or not. This would allow higher factors in higher dimensions, and thus solve the problem of over-fitting.

A second issue to overcome, also visible from the results, is that even informative dimensions can show equal results during some parts of the graph. This can be seen between query number five and ten, in figure 24. In this example only a few queries show similar results, but this raises the question as to how many similar results to allow when judging wether a dimension is informative or not? Most of the training examples were classified by comparing graphs, but perhaps a more in-depth approach should have been used; in cases where there was confusion more than 100 queries was used to try and more accurately describe the instance. What this ultimately comes down to, however, is processing time. The user could select a much larger sample to make the judgement more sound, but at the expense of time.

Most tests show difficulty selecting a factor higher than 30 when number of dimensions exceed three. The tests were mainly done using some of the more populated data sets, where the dimensions would have 1500 to 3000 valid data points. As has been noted earlier, when lowering the factor too much, over-fitting is an issue, and can thus severely limit the use of dimensionality reduction if few data points are present in the data sets being examined.

8 Conclusions and Future Work

The framework developed does show promise with regards to filtering statistical outliers while still maintaining the underlying pattern of the distributions. Even varying selection of noise removal does not seem to affect the detected pattern by a large margin. The suggestion is thus to set the noise removal to a bit higher than expected, and if need be manually adjust it for each query.

Although some generalization of the underlying pattern is made when using projections, the overall results show little over- and under-fitting, irrespective of factor. The choice of factor still remains the most difficult parameter of the framework, and a more elaborate study as to wether this parameter can also be optimized need to be made. Despite still having a parameter which the user will have to fine-tune for each data set, the parameter itself is simple to understand and thus increases the usability of the framework.

The algorithm which determines the number of empty intervals to look for can more than likely be improved upon. At this point the algorithm selects the number of intervals by means of heuristics, and obviously there is a desire to make this process more scientifically sound compared to the current method. The way in which the framework has been constructed, software engineering wise, makes the task of incorporating a new version very easy. This is also true for all other modular parts of the framework; selection of neighbourhood, clustering algorithms and stable width classifier.

Many of the tasks of the framework can be optimized through parallelization. At this point the framework does not make use of multiple cores, which is standard in today's computers. The DiMaxL algorithms can be heavily optimized by use of parallalism given that dynamic programming is used as method to solve the problem. As the framework relies on comparing different instances from DiMaxL, this process can also be optimized by the same means. In the future, the hope is that both algorithms can be converted to c++, while also making the framework operate in a client-server fashion, where a central server stores the solved instances. Using this setup a group of people collaborating can more easily solve a task without having to worry about duplicating each others' work.

The major obstacle for the dimensionality reduction part is missing or lack of values. With enough points, the function seems to be working properly, but one has to be weary of too little data with increased dimensions. It is thus recommended to reduce the dimensions semi-automatically by using a separate factor for each reduction. One solution to this problem is to use a less strict function when deciding if an instance is valid or not.

References

[1] David Baker and Andrej Sali. Protein structure prediction and structural genomics. *Science*, 294:93–96, 2001.

[2] Anders Bergkvist and Peter Damaschke. Fast algorithms for finding disjoint subsequences with extremal densities. *Pattern Recogn.*, 39(12):2281–2292, 2006.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[4] P. S. Bradley and Usama M. Fayyad. Refining initial points for k-means clustering. pages 91–99. Morgan kaufmann, 1998.

[5] Engelbert Buxbaum. *Fundamentals of Protein Structure and Function*. Springer US, 2007.

[6] Jianlin Cheng, A.N. Tegge, and P. Baldi. Machine learning methods for protein structure prediction. *IEEE Reviews in Biomedical Engineering*, 1:41 – 9, 2008//.

[7] R.G. Cowell, A.P. Dawid, T. Hutchinson, and D.J. Spiegelhalter. A bayesian expert system for the analysis of an adverse drug reaction. *Artificial Intelligence in Medicine*, 3(5):257 – 70, 1991/10/. Bayesian expert system;adverse drug reaction;prototype probabilistic expert system;drug-induced pseudomembranous colitis;simple Bayesian networks;probability distributions;.

[8] F.J. Diez, J. Mira, E. Iturralde, and S. Zubillaga. Diaval, a bayesian expert system for echocardiography. *Artificial Intelligence in Medicine*, 10(1):59 – 73, 1997/05/. DIAVAL;Bayesian expert system;echocardiography;heart diseases;diagnosis;causal probabilistic model;knowledge base;Bayesian network;OR gate;a posteriori probabilities;written report;.

[9] Martin Ester, Hans peter Kriegel, JÃ¶rg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[10] M Gerner, Lith A, Mattsson J, and Werner P. Mining for large almost empty regions in data sets implementation, evaluation and analysis of the disjoint intervals of maximum length algorithms. 2007.

[11] Ting Han, Bo Li, and Limei Xu. A universal fault diagnostic expert system based on bayesian network. volume vol.1, pages 260 – 3, Piscaatway, NJ, USA, 2008//. universal fault diagnostic expert system;Bayesian network;sensor data;equipment structure;maintenance decision making support;.

[12] Huang Hanmin, Huang Xiyue, Zhang Ping, Chai yi, and Shi Weiren. Ann-based handwritten character recognition. pages 1177 – 80, Tokyo, Japan, 1999//. ANN-based handwritten character recognition;digital image processing;modified learning factor;adaptation;classification;modified BP neural network algorithm;.

[13] Alexander Hinneburg, Er Hinneburg, and Daniel A. Keim. An efficient approach to clustering in large multimedia databases with noise. pages 58–65. AAAI Press, 1998.

[14] Er Hinneburg and Hans henning Gabriel. Denclue 2.0: Fast clustering based on kernel density estimation. In *In Proceedings of The 7th International Symposium on Intelligent Data Analysis*, pages 70–80, 2007.

[15] Thorsten Joachims, Fachbereich Informatik, Fachbereich Informatik, Fachbereich Informatik, Fachbereich Informatik, and Lehrstuhl Viii. Text categorization with support vector machines: Learning with many relevant features, 1997.

[16] George Karypis, Eui-Hong (Sam) Han, and Vipin Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling, 1999.

[17] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data An Introduction to Cluster Analysis*. Wiley Interscience, New York, 1990.

[18] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[19] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[20] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. pages 144–155, 1994.

[21] Huzefa Rangwala and George Karypis. Profile based direct kernels for remote homology detection and fold recognition. *Bioinformatics*, page bti687, 2005.

[22] Burkhard Rost and Chris Sander. Prediction of protein secondary structure at better than 70accuracy, 1993.

[23] Henry A. Rowley, Student Member, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *IEEE Transactions On Pattern Analysis and Machine intelligence*, 20:23–38, 1998.

[24] H.Y.Y. Sanossian. An arabic character recognition system using neural network. pages 340 – 8, New York, NY, USA, 1996//. Arabic character recognition system;optical character recognition system;multilayer perceptron classifier;input features;invariant character recognition;.

[25] Kurt Wüthrich. Nmr studies of structure and function of biological macromolecules, 2002.

[26] Ming-Hsuan Yang, David J. Kriegman, and Narendra Ahuja. Detecting faces in images: A survey. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 24(1):34–58, 2002.