

This article was downloaded by: [University of Haifa Library]

On: 08 August 2012, At: 14:22

Publisher: Taylor & Francis

Informa Ltd Registered in England and Wales Registered Number: 1072954 Registered office: Mortimer House, 37-41 Mortimer Street, London W1T 3JH, UK



## Molecular Crystals and Liquid Crystals

Publication details, including instructions for authors and subscription information:

<http://www.tandfonline.com/loi/gmcl20>

### Prediction of the Liquid-Crystalline Property Using Different Classification Methods

Florin Leon<sup>a</sup>, Cătălin Lisa<sup>b</sup> & Silvia Curteanu<sup>b</sup>

<sup>a</sup> Department of Computer Science and Engineering, Technical University, "Gh. Asachi," Iași, Romania

<sup>b</sup> Department of Chemical Engineering, Technical University, "Gh. Asachi," Iași, Romania

Version of record first published: 18 Mar 2010

To cite this article: Florin Leon, Cătălin Lisa & Silvia Curteanu (2010): Prediction of the Liquid-Crystalline Property Using Different Classification Methods, *Molecular Crystals and Liquid Crystals*, 518:1, 129-148

To link to this article: <http://dx.doi.org/10.1080/15421400903574391>

PLEASE SCROLL DOWN FOR ARTICLE

Full terms and conditions of use: <http://www.tandfonline.com/page/terms-and-conditions>

This article may be used for research, teaching, and private study purposes. Any substantial or systematic reproduction, redistribution, reselling, loan, sub-licensing, systematic supply, or distribution in any form to anyone is expressly forbidden.

The publisher does not give any warranty express or implied or make any representation that the contents will be complete or accurate or up to date. The accuracy of any instructions, formulae, and drug doses should be independently verified with primary sources. The publisher shall not be liable for any loss, actions, claims, proceedings, demand, or costs or damages whatsoever or howsoever caused arising directly or indirectly in connection with or arising out of the use of this material.

# Prediction of the Liquid-Crystalline Property Using Different Classification Methods

FLORIN LEON,<sup>1</sup> CĂTĂLIN LISA,<sup>2</sup> AND  
SILVIA CURTEANU<sup>2</sup>

<sup>1</sup>Department of Computer Science and Engineering, Technical University  
“Gh. Asachi,” Iași, Romania

<sup>2</sup>Department of Chemical Engineering, Technical University  
“Gh. Asachi,” Iași, Romania

*This article reports a new approach to predict the liquid crystalline behavior using three classes of machine learning algorithm: eager learners (C4.5, random tree, random forest, reduced error pruning tree), lazy learners (k-nearest neighbor, non-nested generalized exemplars with prototypes), and neural networks. The performance analysis for these algorithms supposes calculating the errors both for the training and validation data, in order to determine the accuracy in building the model and its generalization capability. A large database containing 390 compounds with different units connected to the bis- and tris-phenyl aromatic, azo-aromatic, azomethinic types was used as example. The prediction of liquid crystalline property is correlated with chemical structure, type of compound, molecular weight, and a series of structural characteristics estimated by mechanical molecular simulation. Advantages and disadvantages are discussed for each method; the best results are obtained with lazy classification algorithms, especially with our original one, non-nested generalized exemplars with prototypes (NNGEP).*

[Supplementary materials are available for this article. Go to the publisher's online edition of *Molecular Crystals and Liquid Crystals* for the following free supplemental resource: Data Base]

**Keywords** Classification algorithms; liquid crystalline behavior; neural networks

## Introduction

Presently, the reduction of the number of experimental trials represents a requirement that is increasingly felt in the field of the study and analysis of physical, chemical, and biological phenomena. In particular, this tendency is encouraged by the economical advantages that experimental research can get from it in terms of time and money savings. Determination of the liquid crystals properties of some organic compounds based on their structures, quantitative structure–property relationship, is a major research subject in computational chemistry.

---

Address correspondence to Silvia Curteanu, Department of Chemical Engineering, Technical University “Gh. Asachi,” B-dul D. Mangeron No. 71A, 700050, Iași, Romania. E-mail: [silvia\\_curteanu@yahoo.com](mailto:silvia_curteanu@yahoo.com)

The property prediction methods may be evaluated based on their classification as empirical, semi-empirical, theoretical, and hybrid approaches. The empirical methods usually require extensive data collection and result in linear or simple nonlinear structure–property relations. Computations are very rapid at the expense of prediction accuracy. In addition, these methods require a specific functional form that may not always be available and the parameters determined by regression from the data. They are also computationally expensive but provide excellent property estimations. Most approaches settle for the middle ground by utilizing simplified assumptions such as those found in semi-empirical methods and hybrid approaches. These methods provide the best compromise between model development effort, computational time, and property prediction accuracy. In this regard, neural network (NN)-based methods offer advantages of ease of development and implementation and execution speed while maintaining the prediction accuracy. Different machine learning algorithms, including hierarchical clustering, decision trees, nearest neighbors, support vector machines, and bagging can be also used in structure prediction [1].

One of the most interesting properties of compounds is the liquid-crystalline (LC) behavior, because in this state the materials combine two essential properties of the matter: the order and the mobility. But, due to the complexity of the liquid-crystalline phase, it is not at all easy to predict the occurrence of a mesophase. There are many methods of predicting the liquid-crystalline behavior, based on molecular, energetic, or structure–property relationship models [2–9]. Kranz *et al.* [10] presented an example of a new way to predict a property from the chemical structure of a chemically heterogeneous class of compounds. The clearing temperatures of nematic liquid-crystalline phases of a large number of compounds were used to train neural networks to derive this material property directly from their chemical structure.

Our group has made significant contribution to liquid-crystalline behavior prediction using different learning methods, such as neural networks and categorization algorithms. These methods have been applied to various classes of compounds: copolyethers with mesogene groups in the main chain [11], ferrocene derivatives [12,13], or azo-aromatic compounds [14].

In this article we used an organic compounds database with 390 records [15] that includes a wide variety of compounds: bis- and tris-phenyl aromatic, azo-aromatic, azomethinic types containing connecting groups in the rigid core as azo, azomethine, or double bond. We report a new approach to predict the liquid-crystalline behavior for these compounds using neural networks and classification algorithms.

A performance analysis for several well-known classification algorithms is made. The error is calculated for both the training set, to determine how accurately a certain algorithm can build a model of the data, and a validation set, to obtain the prediction capability of that model. The algorithms presented here belong to two important classes of classification methods: eager learners and lazy learners. Eager learners use many computational resources in the first step to build the actual model, and then the prediction is easy; we chose different decision tree inducers for this class: C4.5, random tree, random forest, and REPTree. Lazy learners, on the other hand, build a very simple model, and most of the processing is made in the second step, for prediction; for this category, we chose a set of classifiers based on the nearest neighbor paradigm: simple nearest neighbor,  $k$ -nearest neighbor, and non-nested generalized exemplars with prototypes.

This article follows a paper [11] where the liquid-crystalline behavior for a series of copolyethers with mesogene groups in the main chain has been investigated.

In this previous article, it has been shown that for the studied copolyethers the neural networks prove to be efficient classification tools. In the present article, the actual case study implies the use of classification algorithms for the crystalline property prediction rather than neural networks. An original classification algorithm is also used with accurate prediction results. It extends the nearest neighbor methods by allowing the training instances to be grouped into hyper-rectangles that are treated unitarily during the prediction phase, which can greatly improve the speed of the classification, and also by computing information about the prototype of mode value of the attributes of such hyper-rectangles, which helps to give additional information of an instance regarding how relevant it is to a class, thus allowing a fuzzy logic interpretation of the results.

One should note that different methods have been presented and tested based on classification algorithms and neural networks in order to develop a general methodology, applicable to different processes from the chemical industry. Thus, depending on the application, one can use the most appropriate classification algorithm from the package presented here.

## Classification Techniques

In this section we describe each classification algorithm mentioned above. From the implementation point of view, we used the variants described by Witten and Frank [16] and Leon [17].

Classification is a procedure in which individual items or objects, often referred to as *instances*, are placed into groups, or classes, based on quantitative information on one or more of their characteristics, referred to as *attributes*. Classification is a supervised technique; *i.e.*, the model is built based on a training set of instances whose classes are known. Formally, given the training data  $\{(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)\}$ ,  $\vec{x}_i \in X$  and  $y_i \in Y$ , the goal is to determine a classifier  $h: X \rightarrow Y$  that is a good approximation of the mapping of any object  $\vec{x}_i$  to its classification label, or class  $y_i$ . More simply, the purpose of a classification algorithm is to find a hypothesis  $h$  from examples; *i.e.*, pairs  $(\vec{x}, f(\vec{x}))$ , where  $f$  is a function with a discrete codomain, such that  $h \approx f$  holds not only for the given training pairs but for other arguments  $\vec{x}$  from their domain. Thus the information contained in the training set (with instances whose corresponding class labels are known) is used to classify new, previously unseen instances based on an explicit model (for “eager” learners) or an implicit model (for “lazy” learners).

C4.5 is a software extension of the basic ID3 algorithm designed by Quinlan [18] to address the following issues not dealt with by ID3: avoiding overfitting the data, determining how deeply to grow a decision tree, reducing error pruning, rule post-pruning, handling continuous attributes, choosing an appropriate attribute selection measure, handling training data with missing attribute values, handling attributes with differing costs, and improving computational efficiency [19].

The C4.5 algorithm generates a decision tree for a given data set by recursive partitioning of data. The decision tree is grown using a depth-first strategy [20]. The algorithm considers all the possible tests that can split the data set and selects a test that gives the best information gain. For each discrete attribute, one test with as many outcomes as the number of distinct values of the attribute is considered. For each continuous attribute, binary tests involving every distinct value of the attribute are considered. In order to gather the entropy gain of all these binary tests efficiently,

the training data set belonging to the node in consideration is sorted for the values of the continuous attribute and the entropy gains of the binary cut based on each distinct value are calculated in one scan of the sorted data. This process is repeated for each continuous attribute.

The algorithm supports decision tree pruning at the end of the training process. In our comparison, we test C4.5 both with and without pruning.

The *random tree* (RT) classifier [21] builds a tree that considers  $k$  random features at each node and performs no pruning; therefore, its error rate on the training set alone is very small. A random forest (RF) [22] is composed of several classification trees. To classify a new object from an input vector, the input vector is run down each of the trees in the forest. Each tree gives a classification, and this is considered to be a “vote” for that class. The forest chooses the classification having the most votes over all the trees in the forest.

The *reduced error pruning tree*, REPTree [21], is a fast algorithm that builds a decision tree using information gain or variance reduction and prunes it using reduced-error pruning with back-fitting. It only sorts values for numeric attributes once. Missing values are dealt with by splitting the corresponding instances into pieces, like C4.5.

As stated before, instance-based learning reduces the learning effort by simply storing the examples presented to the learning agent and classifying the new instances on the basis of their closeness to their “neighbors”; *i.e.*, previously encountered instances with similar attribute value. The *nested generalized exemplar* (NGE) theory [23] is an incremental form of inductive learning from examples, which extends the nearest neighbor classification method. NGE is a learning paradigm based on class exemplars, where an induced hypothesis has the graphical shape of a set of hyper-rectangles in an  $n$ -dimensional Euclidean space. Exemplars of classes are either hyper-rectangles or single training instances; *i.e.*, points, known as *trivial hyper-rectangles*.

The input to the NGE system is a set of training examples presented incrementally, each described as a vector of numeric attribute/value pairs and an associated class. The  $n$  attributes used for describing the examples define the  $n$ -dimensional Euclidean space in which the concept will be represented.

NGE generalizes an initial user-defined set of points, or *seeds*, in the  $n$ -dimensional Euclidean space, expanding (or in some special situations shrinking) them along one or more dimensions as new training examples are presented. The choice of which hyper-rectangle to generalize depends on the distance metric. If the attributes have crisp values, such a metric is a weighted Euclidean distance, either point-to-point or point-to-hyper-rectangle.

Wettschereck and Dietterich [24] showed that the performance of NGE is poor on many problems, mainly because of overgeneralization. It was hypothesized that these issues were caused by allowing hyper-rectangles to nest or overlap.

Martin [25] proposed a solution to avoid all forms of overgeneralization by never allowing exemplars to nest or overlap in an algorithm called *non-nested generalized exemplar* (NNGE). This is prevented by testing each prospective new generalization to ensure that it does not cover any negative examples and by modifying any generalizations that are later found to do so. This heuristic permits a reasonable level of generalization while preventing the overgeneralization that would occur if every example were generalized to its nearest neighbor of the same class, even though that exemplar may be far away in attribute space with many intervening negative

examples. Because NNGE prevents overgeneralization by correcting any overlapping or nesting, it always tries to generalize new examples to their nearest neighbor of the same class, but if this is impossible due to intervening negative examples, no generalization is performed. If a generalization later conflicts with a negative example, it is modified to maintain consistency.

The function used to compute the distance from an instance to a hyper-rectangle is common to NGE and NNGE:

$$d(I, H) = w_H \cdot \sqrt{\sum_{i=1}^m \left( w_i \cdot \frac{d_i(I, H)}{\maxval_i - \minval_i} \right)^2} \quad (1)$$

with

$$d_i(I, H) = \begin{cases} I_i - H_{\text{upper}_i}, & \text{if } I_i > H_{\text{upper}_i} \\ H_{\text{lower}_i} - I_i, & \text{if } I_i < H_{\text{lower}_i} \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

where  $w_H$  is the exemplar (hyper-rectangle) weight,  $w_i$  are attribute weight,  $I_i$  is the  $i$ th attribute value of  $I$ ,  $\maxval_i$  and  $\minval_i$  are the upper and lower bounds of the domain on dimension  $i$ , and  $H_{\text{upper}}$  and  $H_{\text{lower}}$  are the upper and lower bounds of the hyper-rectangle  $H$ .

For symbolic attributes,

$$d_i(I, H) = \begin{cases} 0, & \text{if } I_i \in H_i \\ 1, & \text{if } I_i \notin H_i \end{cases} \quad (3)$$

In the variant presented by Witten and Frank [16], NNGE uses the IB4 [26] attribute weighting scheme and mutual information [27] and gives up weighting of exemplars.

Experimental results showed that not all the elements of a category are processed in the same way; some elements are considered more typical of one category than others are. This is known as the *prototypical effect* [28]. For a European, an apple is more representative for the fruit category than a coconut. Cognitive psychology gives the prototype two possible interpretations [29]. It can represent one or more real exemplars, which appear most frequently when human subjects are asked to exemplify a category. The second approach considers the prototype to be an ideal exemplar that sums up the characteristics of more members of the category.

The NNGE model was extended in order to include prototype information about each rule or hyper-rectangle. Because a generalized exemplar contains several instances, it is not necessary for the statistical average of those instances to match the center of the corresponding hyper-rectangle. The prototype may differ from the hyper-rectangle geometric center.

The proposed *non-nested generalized exemplars with prototypes* (NNGEP) is an incremental algorithm, so the prototypes must also be computed incrementally. Adding a new instance to a hyper-rectangle is a particularization of the general case of merging two Gaussian prototypes, with means  $\mu_1$  and  $\mu_2$  and standard deviations  $\sigma_1$  and  $\sigma_2$ , respectively. We understand *merging* to mean computing the mean  $\mu$  and standard deviation  $\sigma$  for a new Gaussian prototype that would have resulted if all

the instances of the two source prototypes had been added to it from the start. The equations for this purpose are (4) and (5), where  $n_1$  and  $n_2$  represent the number of instances included in the two prototypes. The number of instances of the resulting prototype is  $n = n_1 + n_2$ .

$$\mu = \frac{n_1 \cdot \mu_1 + n_2 \cdot \mu_2}{n_1 + n_2} \quad (4)$$

$$\sigma = \sqrt{\frac{n_1 \cdot \sigma_1^2 + n_2 \cdot \sigma_2^2}{n_1 + n_2} + \frac{n_1 \cdot n_2 \cdot (\mu_1 - \mu_2)^2}{(n_1 + n_2)^2}} \quad (5)$$

When adding only one new instance to a hyper-rectangle, we can consider it to be a prototype with a mean equal to its coordinates and a null standard deviation. Therefore, the existing hyper-rectangle prototype defined by mean  $\mu_E$  and standard deviation  $\sigma_E$  is updated as shown by Eqs. (6) and (7):

$$\mu'_E = \frac{\mu_E \cdot n_E + \mu_I}{n_E + 1} \quad (6)$$

$$\sigma'_E = \frac{\sqrt{n_E \cdot (n_E + 1) \cdot \sigma_E^2 + n_E \cdot (\mu_E - \mu_I)^2}}{n_E + 1} \quad (7)$$

This helps computing the similarity between a given instance and the closest generalized exemplar. Although classification can be based on similarity, the two notions are not identical. Even if an instance has been assigned a certain category due to an existing rule, it may not be representative of that category. An important advantage of the model built by NNGEP is that instances have graded memberships to categories, which permit a fuzzy interpretation using fuzzy numbers with multidimensional Gaussian membership functions.

The above procedure can only be applied to numeric attributes. When attributes are symbolic or discrete, their mean value cannot be computed. Instead, the mode of the set of attribute values is computed; *i.e.*, the value that occurs most frequently in the given set. The distance on a certain dimension between an instance and a prototype is either 0 if the instance attribute value on that specific dimension is the same as the mode, or 1 otherwise.

*Feed-forward neural networks* are a method for building models when a non-linear relationship is assumed [30]. The processing elements of a network, the neurons, are organized in layers and each neuron is linked to the neurons of the next layer. Typically, a feed-forward network consists of one input layer, some hidden layers, and an output layer. In the training phase, the neural network learns the behavior of the process. The training data set contains both input patterns and the corresponding output patterns, also called *target patterns*. Neural training leads to finding values of connection weights that minimize differences between the network outputs and the target values. The most extensively adopted algorithm for the learning phase is the back-propagation algorithm. The purpose of developing a neural model is to devise a network that captures the essential relationships in the data. Then it can be applied to new sets of inputs to produce corresponding outputs. This is called *generalization* and represents the subsequent phase after training; *i.e.*, validation phase. A network is said to generalize well when the input-output

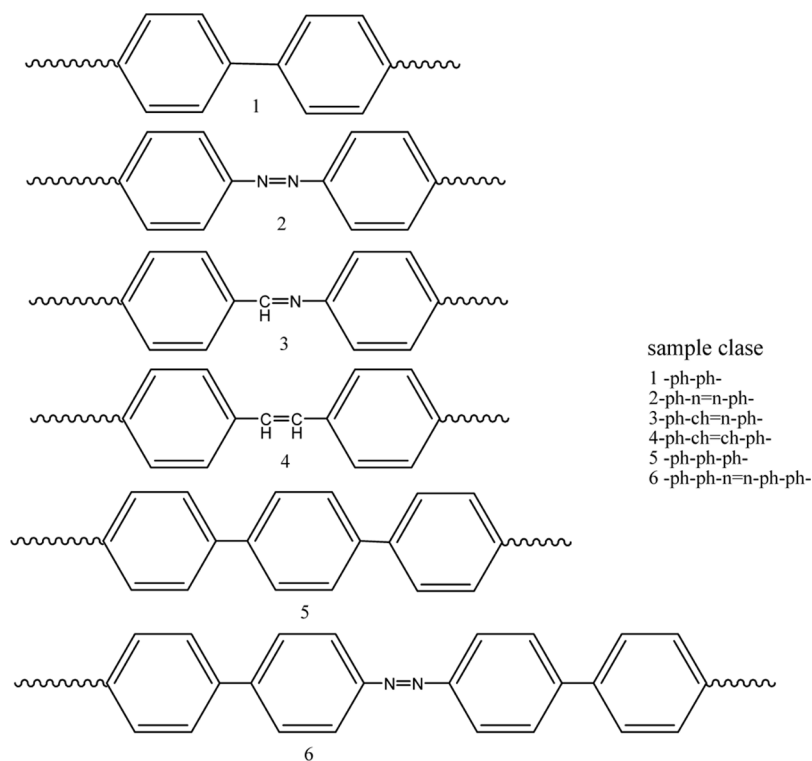
relationship found by the network is correct for the input–output patterns of validation data that were never used in training the network (unseen data).

In practical application for materials research, it is necessary to prepare a large enough database of experimental results to develop an neural network with a good performance. The architecture, transfer function, training algorithm, and other parameters of the neural network should be carefully set and modified during optimization. Thus, the well-trained neural network can be used to obtain the solutions of new input data in the same domain of the experimental database. This process can be summarized in the following four stages:

1. Collect and preprocess the experimental data.
2. Train the network and optimize its configuration.
3. Evaluate the performance of the network; return to stage 2 if the performance is not satisfactory.
4. Use the trained network for simulation or prediction.

### Experimental Arrangement

The combination of different structural units in a molecule gives rise to physical properties that are very important when designing new liquid crystals. For practical use, the materials should not only have the molecular structure suitable for inducing liquid crystal properties but an appropriate combination of physical properties for



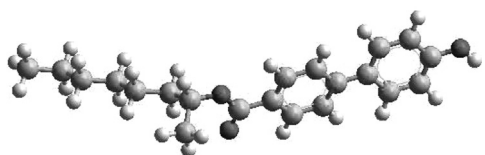
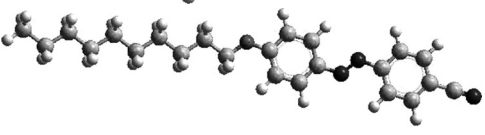
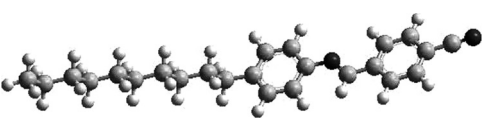
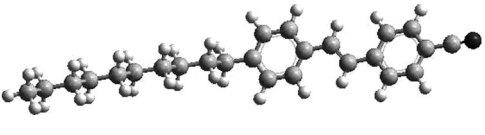
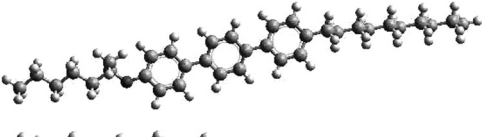
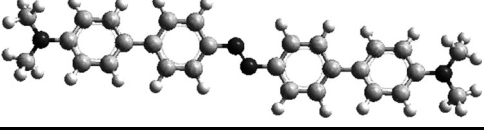
**Figure 1.** The general structures of the analyzed compounds.



that application. The factors influencing the molecular unit are varied and include core units, connecting groups, terminal groups, lateral groups, and lengths of flexible chains. All these structural factors affect the nature of interactions between liquid-crystalline molecules and are very important for obtaining the adequate mesomorphic behavior. The organic compounds used in this article have similar structures with small structural changes that allow a systematical analysis of the factors that influence liquid crystal properties and determination of some parameters that will be used in prediction with neural networks. Our database contains organic compounds with different units connected to the bis- and tris-phenyl aromatic, azo-aromatic, and azomethinic structures, containing connecting groups as azo, azomethine, or double bond (Fig. 1). Figure 1 presents the classes of our database noted 1 to 6.

The establishment of the numerical inputs for machine learning models (*molecular descriptors*) is a critical and difficult problem. This is due to the fact that the molecular descriptors must represent the molecular structural features related to the properties of interest as distinctly as possible. The accurate prediction of the learning methods depends heavily on the amount of correction between the molecular descriptors and the structural features. We used as topological and geometrical

**Table 1.** Values of 3D and molecular descriptors

Sample class	Three-dimensional (3D) structure for several organic compounds	$L_{rig}$ (Å)	$L_{flex}$ (Å)	S	M (a.m.u.)
1		10.74	8.08	0.146	326.4
2		11.78	13.95	0.110	363.5
3		11.88	12.76	0.120	346.5
4		12.04	12.76	0.119	345.5
5		11.52	21.56	0.089	484.7
6		17.89	5	0.126	420.5

molecular descriptors: length of the rigid core ( $L_{\text{rig}}$ ), length of the flexible core ( $L_{\text{flex}}$ ), molecular weight ( $M$ ), and an asymmetry factor calculated as ratio of molecular diameter/total length ( $S$ ).

The molecular modeling studies (molecular mechanics) were carried out using the HyperChem 7.5 software package. Submitting a structure to a calculation can be expensive in terms of human time and effort. HyperChem lets us build and display molecules easily. Because HyperChem contains a graphical interface, we can monitor the construction of molecules. Using the Drawing tool, we can draw a two-dimensional (2D) representation of a molecule and then use the Model Builder to generate a three-dimensional (3D) structure. The Model Builder adds implicit hydrogens to the molecule at our request. We can also manipulate individual bonds, bond geometries, angles, torsions, and atomic charges during model building or after model building. HyperChem contains a database of amino and nucleic acid residues so we can quickly build compounds containing these subunits.

To calculate the molecular descriptors [30], we need to generate a well-defined structure. A calculation often requires a structure that presents a minimum on a potential energy surface. HyperChem contains several geometry optimizers to do this (Polak-Ribière, steepest descendent, or Fletcher-Reeves [conjugate gradient]), optimizer RMS (gradient) of  $0.01 \text{ kcal}/(\text{mol} \cdot \text{\AA})$ . We can then calculate single point properties of a molecule or use the optimized structure as a starting point for subsequent calculations, such as molecular dynamics simulations.

The molecular mechanics (MM+) force field was applied for preliminary structure optimization and study of the conformational behavior of each organic compound. Molecular mechanics has been shown to produce more realistic geometry values for the majority of organic molecules because it is highly parameterized [31].

The experimental results of HyperChem calculation for the several analyzed organic compounds are shown in Table 1.

## Results and Discussion

The present work uses classification algorithms and neural networks to predict the liquid-crystalline behavior of some organic compounds with different units connected to the bis- and tris-phenyl aromatic, azo-aromatic, and azomethinic structures, containing connecting groups as azo, azomethine, or double bond.

In the following sections, we apply the classification algorithms previously described. For each category of tests, the average,  $\rho$ , and the standard deviation,  $\sigma$ , are computed. In the tables that present the results, the best experiment, with the lowest error, is emphasized.

Input data are the five parameters presented in Table 1: length of the rigid core ( $L_{\text{rig}}$ ), length of the flexible core ( $L_{\text{flex}}$ ), molecular weight ( $M$ ), ratio of molecular diameter and total length ( $S$ ), and compound class (classcomp) (1–6). Concerning the liquid crystal behavior, we have coded the possibility to generate a mesophase with 1 and the crystalline or amorphous phases with 0. This is the symbolic output of the model.

### Nearest Neighbor

First, the training set error was considered. Also, the prediction capabilities of the model are estimated with two methods. For cross-validation, the data set is divided

**Table 2.** Classification results using the Nearest Neighbor algorithm

No.	Training set error	Cross-validation error ( $n = 10-20$ )	Data set split error (2/3-1/3)
1	0	18.557	15.909
<b>2</b>	<b>0</b>	<b>17.784</b>	<b>15.385</b>
3	0	17.784	15.385
4	0	17.784	15.541
5	0	18.299	16
$\rho/\sigma$	0/0	18.042/0.333	15.644/0.267

The best prediction result is marked with bold.

into  $n$  parts, and the following procedure is repeated  $n$  times:  $n - 1$  parts are used as a training set, and the  $n$ th part is used as a validation set. The final error represents the average error of the  $n$  trials. Then, the data set is split into two thirds for training and one third for validation and the error is computed only for the validation set. The results of the tests are displayed in Table 2. The best prediction result is marked with bold. The main characteristic of a nearest-neighbor model is that, in the absence of noise, the error rate on the training set is 0. The prediction error, which is important to estimate the utility of the method, has an acceptable value of around 15%.

### **k-Nearest Neighbor**

For the tests presented in Table 3, we vary the number of neighbors,  $k$ , and we weight the importance of the neighbors by the inverse of the distance. Because the data set split method is (more than  $n$  times) faster than cross-validation, in the following we only compute the training set error and data set split error. When  $k$  increases, there is no longer a guarantee that the error on the training set will be 0. However, the prediction error can be lower than in the case of simple NN.

Because the lowest error is achieved when there are four neighbors considered, for the following tests, whose results are displayed in Table 4, we keep  $k = 4$  and test different metrics for weighting.

**Table 3.** Classification results obtained with the  $k$ -Nearest Neighbor algorithm using inverse distance weighting and varying the number of neighbors

$K$	Training set error	Data set split error (2/3-1/3)
1	0	16.279
2	0	16.279
3	8.505	17.054
<b>4</b>	<b>6.701</b>	<b>14.729</b>
5	10.825	16.279
$\rho/\sigma$	5.206/4.539	16.124/0.775

The best prediction result is marked with bold.

**Table 4.** Classification results obtained with the  $k$ -Nearest Neighbor algorithm using different distance weighting metrics

Metric	Training set error	Data set split error (2/3–1/3)
<b>1/distance</b>	<b>6.701</b>	<b>14.729</b>
No weighting	12.629	17.829
1-distance	8.505	14.729
$\rho/\sigma$	9.278/2.632	15.762/1.550

The best prediction result is marked with bold.

For our problem, it seems that the best results for nearest neighbor classification are achieved with four neighbors and inverse distance weighting of the neighbors.

### C4.5

For this algorithm we provided two variants, by pruning or not the decision tree obtained. Pruning the decision tree supposedly helps the prediction capability of the model, at the expense of a slight reduction in the accuracy on the training set. First, we consider the unpruned tree. We vary the minimum number of instances that can be found in a leaf node, and the test results are presented in Table 5. In this case, the error rates on both training set and prediction are larger than those obtained with the previous methods because the decision tree techniques are based on a

**Table 5.** Classification results using C4.5 algorithm with unpruned decision tree

Minimum instances per leaf	No. test	Training set error	Data set split error (2/3–1/3)
1	1	10.825	20.513
	2	10.825	20.946
	3	10.825	21.429
	4	10.825	23.485
	5	10.825	16.239
	$\rho/\sigma$	10.825/0	20.552/2.421
2	1	10.309	22.481
	2	10.309	22.481
	3	10.309	20.139
	4	10.309	20.513
	<b>5</b>	<b>10.309</b>	<b>16.239</b>
	$\rho/\sigma$	10.309/0	20.371/2.330
3	1	10.567	21.159
	2	10.567	20.139
	3	10.567	22.727
	4	10.567	21.600
	5	10.567	16.239
	$\rho/\sigma$	10.567/0	20.373/2.273

The best prediction result is marked with bold.

**Table 6.** Classification results using C4.5 algorithm with pruned decision tree while varying the confidence factor

Confidence factor	Training set error	Data set split error (2/3–1/3)
0.10	14.175	21.705
0.20	13.402	21.705
0.25	13.402	21.705
0.30	11.856	21.705
<b>0.50</b>	<b>10.825</b>	<b>21.705</b>
$\rho/\sigma$	12.732/1.240	21.705/0

The best prediction result is marked with bold.

different principle. However, the decision tree results are easier to understand because they are provided in the form of explicit rules.

When we prune the decision tree, we keep the minimum number of instances per leaf  $m_l = 2$ , because the best results on the unpruned tree were obtained in this configuration, and we vary the confidence factor: the smaller the confidence factor, the greater the pruning, which means that the tree will have fewer nodes. The results of the experiments are displayed in Table 6. One can notice that the prediction error rate is worse than that of the unpruned tree for our specific problem.

Next, we consider the confidence factor  $c_f = 0.25$  and we vary the minimum number of instances per leaf, this time on the pruned decision tree. The results are presented in Table 7. The best results are obtained with two minimum instances per leaf.

We further test the tree pruning technique based on reduced error. In this case, a non-terminal node is replaced with a leaf node whose class is that of the majority of its instances, unless the error increases over an allowed threshold by this procedure. The results are presented in Table 8 and show that the error rates are better in case of the reduced error approach.

From these experiments, we can conclude that decision tree pruning does not lead to a better solution for our problem, either for the training set itself or for prediction. However, the reduced error can lead to a slight improvement in the accuracy of the model. Still, the unpruned tree model yields the best results.

**Table 7.** Classification results using C4.5 algorithm with pruned decision tree while varying the minimum number of instances per leaf

Minimum instances per leaf	Training set error	Data set split error (2/3–1/3)
1	13.918	21.705
<b>2</b>	<b>13.402</b>	<b>21.705</b>
3	13.402	21.705
4	13.918	21.705
5	13.660	21.705
$\rho/\sigma$	13.660/0.236	21.705/0

The best prediction result is marked with bold.

**Table 8.** Classification results using C4.5 algorithm with or without reduced error

Reduced error	Training set error	Data set split error (2/3–1/3)
Yes	<b>12.887</b>	<b>20.930</b>
No	13.402	21.750

The best prediction result is marked with bold.

*Random Tree*

For the random tree algorithm, we vary the number of attributes, randomly selected, which are simultaneously considered to create a test. The results are displayed in Table 9.

One can notice that, unlike the C4.5 decision tree, which only considers one attribute for testing at a time, by simultaneously considering two or three attributes for a test, the random tree algorithm can achieve a 0 error on the training set and a slightly lower prediction error.

*Random Forest*

A random forest is composed of a set of random trees, each representing a vote for the class to which the new instance belongs. In this case, we vary the number of trees in the forest. The results are presented in Table 10.

Once the number of trees is increased, the error gets better, although the execution time of the algorithm increases. The best prediction is better, but not significantly, compared to that achieved by the best single random tree.

**Table 9.** Classification results using the random tree algorithm

No. attributes	No. test	Training set error	Data set split error (2/3–1/3)
1	1	0.733	24.806
	2	0.733	21.705
	3	0.516	17.054
	4	0.516	20.155
	5	0.258	24.031
	$\rho/\sigma$	0.567/0.197	21.550/2.848
2	1	0	24.806
	2	0	20.155
	3	0	21.705
	4	0	18.605
	5	0	21.705
	$\rho/\sigma$	0/0	21.395/2.099
3	1	0	17.054
	2	0	19.380
	3	0	18.605
	4	0.516	17.054
	<b>5</b>	<b>0</b>	<b>16.279</b>
	$\rho/\sigma$	0.103/0.211	17.674/1.163

The best prediction result is marked with bold.

**Table 10.** Classification results using the random forest algorithm

No. trees	No. test	Training set error	Data set split error (2/3–1/3)
10	1	0.258	19.380
	2	0.258	16.279
	3	0.516	20.930
	4	0.258	17.829
	5	0	18.605
	$\rho/\sigma$	0.258/0.167	18.605/1.582
20	1	0	18.605
	2	0	17.054
	3	0.258	17.829
	4	0	17.054
	5	0	17.054
	$\rho/\sigma$	0.052/0.105	17.519/0.633
50	1	0	17.829
	2	0	17.054
	3	0	17.829
	4	0	16.279
	<b>5</b>	<b>0</b>	<b>15.504</b>
	$\rho/\sigma$	0/0	16.899/0.922

The best prediction result is marked with bold.

### **REPTree**

Similarly to the C4.5 algorithm experiments, two variants were considered for REPTree: with or without decision tree pruning. The results are displayed in Table 11.

From this table we can again conclude that the pruning affects the accuracy on the training set. However, with an unpruned tree, both the error on the training set and for prediction are comparable to those obtained by random trees.

**Table 11.** Classification results using the REPTree algorithm

Pruning	No. test	Training set error	Data set split error (2/3–1/3)
Yes	1	14.949	20.930
	2	16.753	18.605
	3	11.340	21.705
	4	11.856	18.605
	5	12.887	18.605
	$\rho/\sigma$	13.557/2.062	19.690/1.379
No	<b>1</b>	<b>3.866</b>	<b>18.605</b>
	2	3.866	18.605
	3	3.866	18.605
	4	3.866	18.605
	5	3.866	18.605
	$\rho/\sigma$	3.866/0	18.605/0

The best prediction result is marked with bold.

**NNGEP**

The non-nested generalized exemplars with prototypes finds 62 hyper-rectangles for our problem. A selection of these generalized exemplars is presented in the following:

Class 0 (55 instances)

$L_{fix}$ : 7.1600 11.5200 Prototype: 9.7678 (0.7904)

$L_{flex}$ : 18.9500 29.1000 Prototype: 23.7842 (2.4394)

$s$ : 0.0816 0.1519 Prototype: 0.0957 (0.0138)

mass: 473.0580 564.9000 Prototype: 502.0334 (22.0793)

Classcomp: 1 5 ? Mode: 1 (33)

Class 0 (42 instances)

$L_{fix}$ : 6.6400 14.0900 Prototype: 9.7318 (0.8361)

$L_{flex}$ : 9.5000 19.3200 Prototype: 15.1343 (2.4784)

$s$ : 0.1024 0.1756 Prototype: 0.1238 (0.0171)

mass: 368.5860 410.6670 Prototype: 392.0899 (12.3830)

Classcomp: 1 2 5 Mode: 1 (39)

Class 0 (30 instances)

$L_{fix}$ : 7.1700 11.0600 Prototype: 9.5292 (0.8129)

$L_{flex}$ : 1.5300 12.2900 Prototype: 7.7507 (2.7545)

$s$ : 0.1431 0.2195 Prototype: 0.1668 (0.0199)

mass: 229.3460 347.3040 Prototype: 303.4443 (29.6636)

Classcomp: 1 2 Mode: 1 (29)

Class 0 (28 instances)

$L_{fix}$ : 9.2200 11.5300 Prototype: 10.0471 (0.4573)

$L_{flex}$ : 25.5600 36.7800 Prototype: 31.2018 (3.2645)

$s$ : 0.0674 0.0811 Prototype: 0.0750 (0.0043)

mass: 508.7920 663.0900 Prototype: 571.9829 (44.5216)

Classcomp: 1 2 5 Mode: 1 (25)

...

Class 0 (1 instance)

$L_{fix}$ : 9.2100

$L_{flex}$ : 20.6300

$s$ : 0.0950

mass: 458.6060

Classcomp: ?

There are two kinds of hyper-rectangles that can be observed in the results. For example, the first hyper-rectangle contains several individual instances and can be interpreted as follows: if a new instance has  $L_{fix}$  between 7.16 and 11.52,  $L_{flex}$  between 18.95 and 29.1, etc., and classcomp is 1, 5, or unknown, then the *class* of the new instance is 0. The average value for the  $L_{fix}$  attribute is 9.7678, and the standard deviation of the  $L_{fix}$  attribute values in the training instances is 0.7904. Therefore, if the new instance has the  $L_{fix}$  value close to 9.7678, it not only belongs to the class range of that attribute but is representative for it.



The second type is the trivial hyper-rectangle, composed of a single individual instance. The last hyper-rectangle in the above example belongs to this category. In this case, the hyper-rectangle acts as a regular instance in a nearest-neighbor model, helping to classify new instances only by their distances to existing instances. Information about prototype or mode value is irrelevant here. However, such a trivial hyper-rectangle can transform into a nontrivial one if new instances are added to the model, because the algorithm allows incremental learning of new information.

The training set error is 0%. The error on data set split into two thirds/one third is 14.729%. The algorithm also determines the weights of the attributes for the problem of classifying liquid crystals:

$$w_{L_{\text{fix}}} = 7.635\%$$

$$w_{L_{\text{flex}}} = 15.116\%$$

$$w_s = 12.086\%$$

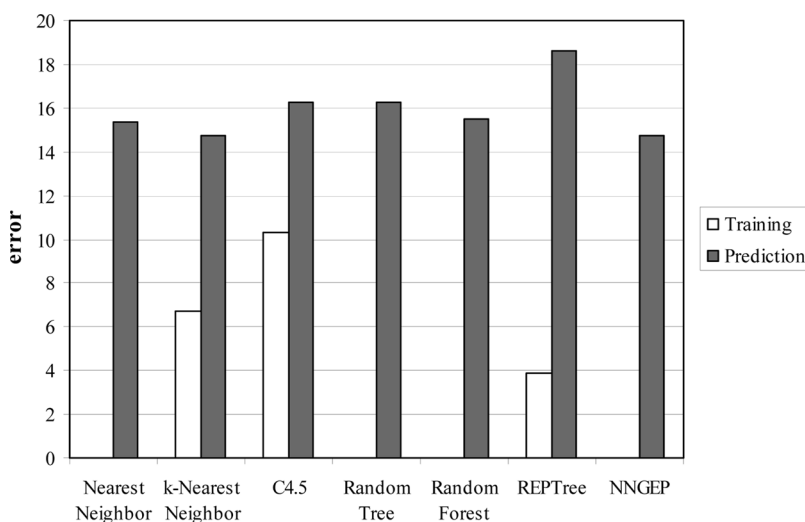
$$w_{\text{mass}} = 15.197\%$$

$$w_{\text{classcomp}} = 49.966\%$$

In other words, the attribute classcomp has the most important role in the decision over the membership of an instance to a class (whether it is a liquid crystal or not).

A comparison between the algorithms, taking into account the best error found, is displayed in Fig. 2.

The instance-based algorithms such as nearest neighbor and NNGETP have zero error for the training set, which is always true unless there are class assignment errors



**Figure 2.** Accuracy comparison between classification algorithms.

**Table 12.** Prediction accuracy for different domains of validation data

Algorithm	33%	20%	10%
Nearest neighbor	84.615	90.67576	95.33788
<i>k</i> -Nearest neighbor	85.271	88.43355	90.86627
C4.5	83.761	86.09706	87.89403
Random tree	83.721	90.13394	95.06697
Random forest	84.496	90.60364	95.30182
REPTree	81.395	87.20127	91.66764
NNGEP	85.271	91.07333	95.53667

in the training set itself; *i.e.*, the same instance assigned more than once with different classes. The *k*-nearest neighbor (with  $k=4$  in our case) has a greater error on the training set and no significantly better prediction accuracy. NNGEP has a good prediction rate, and it also has the advantage of an explicit model compared to the nearest neighbor variants, where all the instances are memorized.

From the decision tree algorithms, random forest and random tree also have zero error for the training set. The prediction error is slightly larger for the random tree; however, its main advantage is that the result can be easily interpretable, unlike the random forest. C4.5 builds a more compact decision tree; however, the errors are not better than those of the random tree and random forest. REPTree has greater errors for both training set and prediction.

The results in Fig. 1 show that the smallest errors in the validation phase, 14.729%, correspond to *k*-nearest neighbor and to our original algorithm NNGEP, associated with 6.701% and 0% in the training phase, respectively. These values mean a prediction accuracy of 85.271% for liquid-crystalline behavior under the condition of a large set of validation data (one third from the whole database). If the domain of the validation (unseen) data becomes narrower, the probability of a correct answer increases, as it can be seen in Table 12.

Of course, our interest is to appreciate the accuracy of the results for a large validation set. In this respect, the proposed algorithm, NNGEP, provides one of the best results.

### Neural Networks

Neural network-based models are relatively model free, in the sense that the underlying functional form is not as rigorous as in the traditional model-based methods. This adds to the generality of these methods.

The use of neural networks to the prediction of properties of organic compound has as a main advantage the fact that neural networks can simulate the nonlinear relationship between structural information and properties of compounds during the training process and generalize the knowledge among homologous series without need for theoretical formulas. The ability of artificial neural networks is significant in determining the quantitative structure–property relationship, because compounds with known properties can be used to train the networks, so that, subsequently, properties of other compounds that cannot be ascertained by experimentation can be determined.

One major problem in the development of a neural network model is represented by the determination of the network architecture; *i.e.*, the number of hidden layers and the number of neurons in each hidden layer to lead to a minimum error. First, potentially good topologies must be identified. Nevertheless, there is no good theory or rule for establishing the best neural network topology that should be used. Therefore, trial-and-error is still required. In this work, the number of hidden layers and units was established by training a different range of networks and selecting the one that best balanced generalization performance against network size.

The neural network with the best performance was MLP(5:42:14:1), a multilayer perceptron with five inputs ( $L_{\text{rig}}$ ,  $L_{\text{flex}}$ ,  $M$ ,  $S$ , classcomp), two hidden layers with 42 and 14 hidden neurons, respectively, and 1 output (with the value of 1 for liquid-crystalline behavior and 0 otherwise).

Although the database has a large number of data, the network predictions are affected by significant errors, both in the training and validation phases. Thus, in this case, the other classification algorithms prove to be more effective, unlike the application described in a previous article [11], where neural networks were the preferred classification tools, both from the point of view of the results provided and from the point of view of the methodology applied.

Acceptable results are obtained in the case of the database with aromatic compounds if the classes of compounds are considered separately. For example, for the class of compounds marked as 2 in Table 1, an MLP(4:42:14:1) network was designed, with MSE (mean squared error) = 0.01831,  $r$  (correlation between experimental data and network predictions) = 0.9785, and  $E_p$  (percentage error) = 3.1133% on the training set. The data in Table 13 show the efficiency of the neural model, which has a probability of a correct answer of 83.33% compared to validation data set (which represents 10% from the data set of compounds 2). Shaded cells represent wrong predictions of the network. This percentage cannot be compared with the results from Table 12 because it is limited to a single class of compound. For the whole database, the probability was less than 75%. In addition, the structure of the neural categorization model is implicit, as it is given by the connection weights.

**Table 13.** The validation phase for MLP(4:42:14:1)

$L_{\text{fix}}$	$L_{\text{flex}}$	$S$	$M$	LC experimental	LC network
9.21	25.5	0.08	463	0	0
9.22	20.98	0.09	439	0	0
9.22	6.22	0.19	270	1	0
9.22	8.77	0.16	298	1	1
9.23	8.9	0.16	296	1	1
9.23	16.62	0.11	381	0	0
9.21	6.39	0.18	266	0	0
9.21	9.94	0.15	310	1	1
9.21	20.61	0.10	439	1	1
9.21	11.69	0.14	360	1	0
9.21	17.24	0.11	431	0	0
9.21	15.2	0.12	404	0	0

Shaded cells represent wrong predictions of the network.

Also, there are many free parameters involved in neural networks, such as choosing the best topology, the learning rate, and the momentum factor to accelerate the learning process, which is rather slow.

The mean squared error was computed using the following formula:

$$MSE = \left( \sum_{j=1}^P \sum_{i=1}^N (d_{ij} - y_{ij})^2 \right) / (N \cdot P) \quad (8)$$

where  $P$  is the number of output processing elements (in this case,  $P = 1$ ),  $N$  is the number of exemplars in the data set,  $y_{ij}$  is the network output for exemplar  $i$  at processing element  $j$ , and  $d_{ij}$  is the desired output for exemplar  $i$  at processing element  $j$ . Smaller errors indicate potentially good architectures; *i.e.*, neural network topologies with chances to train well and to output good results.

All the above used classification algorithms represent good methods for property prediction, but the efficiency of each can be tested and evaluated as a function of a specific problem.

## Conclusions

Different machine learning algorithms—nearest neighbor,  $k$ -nearest neighbor, C4.5, random tree, random forest, REPTree, NNGEP, and neural networks—were used to predict the crystalline behavior for a large database containing compounds with different units connected to the bis- and tris-phenyl aromatic, azo-aromatic, azomethinic types containing connecting groups in the rigid core as azo, azomethine. A series of molecular descriptors that represent the molecular structural features was calculated by molecular modeling: length of the rigid core, length of the flexible core, molecular weight, and an asymmetry parameter.

The mentioned methods were tested in different variants and their efficiency was evaluated based on the error calculated for the training set, to determine how accurately a certain algorithm can build a model of the data, and for various validation sets, to obtain the prediction capability of that model.

The “No free lunch” [32] theorem demonstrates that no heuristic is better than another on the set of all possible problems. The purpose of the tests presented in this article is to give an intuition about the type of classification algorithm that is the most appropriate for the specific problem of classifying liquid-crystalline behavior.

Although neural networks are a popular classification tool in many domains, according to our study, they yielded the worse results for our particular problem. In this case, the best predictions were given by our original algorithm NNGEP, along with perfect accuracy on the training set. It thus combines the good performance of the classic instance-based methods with the lower memory requirements due to its hyper-rectangles and the ease of the interpretation of its explicit model in the form of rules.

## References

- [1] Kleinoder, T., Spycher, S., & Yan, A. (2003). Prediction of properties of compounds. In: *Chemoinformatics—A Textbook*. Gasteiger, J. & Engel, T. (Eds.), Wiley-VCH, Verlag: Weinheim, Germany, 487–622.
- [2] Maier, W. A., & Saupe, Z. (1959). *Naturforsch, A* 14, 882.

- [3] de Gennes, P. G., & Prost, J. (1993). *The Physics of Liquid Crystals*, 2nd ed., Clarendon: Oxford.
- [4] Sen, A. K., & Sullivan, D. E. (1987). *Phys. Rev.*, A 35, 1391.
- [5] de Gennes, P. G. (1979). *Scaling Concepts in Polymer Physics*, Cornell University Press: Ithaca, NY.
- [6] Doi, M., & Edwards, S. F. (1986). *The Theory of Polymer Dynamics*, Clarendon Press: Oxford.
- [7] Jain, S., & Nelson, D. (1996). *Macromolecules*, 29(26), 8523.
- [8] LiqCryst 3.4, Database of Liquid Crystalline Compounds, *Liquid Crystal Group Hamburg*. <http://dwb.unl.edu/Teacher/NSF/C01/C01Links/liqcryst.chemie.uni-hamburg.de/lc/>
- [9] Piotto, S. P. (2002). CURVIS, Bending Energy Calculator, 2nd release, ETH Zürich.
- [9] Piotto, S. P. (2002). FRACTALS, Numerical Curvature Evaluation, ETH Zürich (Software) <http://www.polymers.unisa.it/members/piotto/about.htm>
- [10] Kranz, H., Vill, V., & Meyer, B. (1996). *J. Chem. Inform Sci.*, 36, 1173.
- [11] Leon, F., Curteanu, S., Lisa, C., & Hurdac, N. (2007). *Mol. Cryst. Liq. Cryst.*, 469, 1.
- [12] Lisa, C., Curteanu, S., Bulacovschi, V., & Apreutesei, D. (2008). *Rev. Roum. Chim.*, 53(4), 283.
- [13] Lisa, C., Curteanu, S., Lisa, G., & Apreutesei, D. (2007). *Bulletin of the Transilvania University of Braşov*, 3, 521.
- [14] Lisa, C., & Curteanu, S. (2007). *Comp. Aided Chem. Eng.*, 24, 39.
- [15] LiqCryst Online, Liquid Crystal Group Hamburg, <http://liqcryst.chemie.uni-hamburg.de/en/lolas.php>
- [16] Witten, I. H., & Frank, E. (2000). *Data Mining: Practical Machine Learning Tools with Java Implementations*, Morgan Kaufmann: San Francisco.
- [17] Leon, F. (2006). *Intelligent Agents with Cognitive Capabilities*, Tehnopress: Iaşi, Romania.
- [18] Quinlan, R. (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers: San Mateo, CA.
- [19] Hamilton, H., Gurak, E., Findlater, L., & Olive, W. (2002). *Knowledge Discovery in Databases*, University of Regina: Canada. <http://www2.cs.uregina.ca/~hamilton/courses/831/notes/ml/dtrees/c4.5/tutorial.html>
- [20] Joshi, K. P. (1997). *Analysis of Data Mining Algorithms*. [http://userpages.umbc.edu/~kjoshi1/data-mine/proj\\_rpt.htm](http://userpages.umbc.edu/~kjoshi1/data-mine/proj_rpt.htm)
- [21] Dietterich, T. G. (2000). *An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization*, *Machine Learning*, Springer: Netherlands, Vol. 40(2), 139–157.
- [22] Breiman, L. (2001). *Machine Learning*, 45(1), 5.
- [23] Salzberg, S. L. (1991). *Machine Learning*, 16, 251.
- [24] Wettschereck, D., & Dietterich, G. (1994). *Machine Learning*, 19(1), 5.
- [25] Martin, B. (1995). *Instance-Based Learning: Nearest Neighbour with Generalisation*, Master of science thesis, University of Waikato, Hamilton, New Zealand.
- [26] Aha, D. (1992). *Int. J. Mach. Studies*, 36, 267.
- [27] Shannon, C. E. (1948). *Bell Syst. Tech. J.*, 27, 379.
- [28] Miclea, M. (1999). *Cognitive Psychology*, Polirom: Iaşi, Romania.
- [29] Medin, D. L., Ross, B. H., & Markman, A. B. (2005). *Cognitive Psychology*, 4th edition, John Wiley & Sons, USA.
- [30] Todeschini, R., & Consonni, V. (2000). *Handbook of Molecular Descriptors*, Wiley-VCH: Weinheim, Germany.
- [31] Young, D. (2001). *Computational Chemistry: A Practical Guide for Applying Techniques to Real World Problems*, John-Wiley & Sons: New York.
- [32] Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1, 67.