An Optimized Modularity-Based High Level Classification Model

1st Tiago Colliri
Dept. of Computer Science
ICMC - University of Sao Paulo
Sao Carlos, Brazil
tcolliri@usp.br

2nd Weiguang Liu
School of Computer Science
Zhongyuan University of Technology
Zhengzhou, China
weiguang.liu@zut.edu.cn

3rd Liang Zhao
Dept. of Computing and Mathematics
FFLCRP - University of Sao Paulo
Ribeirao Preto, Brazil
zhao@usp.br

***Abstract*—In this paper, we introduce a network-based classification model which, instead of mapping each data instance as a node in a network, as usual, it maps each data instance attribute as being a node. This procedure allows the model to preserve more information from the input dataset when building the network, specially for datasets with a larger number of features, and thus to make use of this extra information during the training phase. In addition, we also introduce a technique intended to generate a network with one component per class in the dataset while keeping the threshold parameter, which is responsible for determining the edges among the nodes, at a minimum value. In this way, the network emerging from this process is more sensitive to the insertion of new instances, during the testing phase, in terms of its modularity measure, which allows the classifier to infer the new labels based mainly on this measure. We evaluate the model by applying it to both artificial and real benchmark classification datasets, and have its performance compared to those obtained by other traditional classification models on the same data. The preliminary results are encouraging, with the proposed model being ranked on second place among the 10 classifiers considered, on the selected datasets.**

***Index Terms*—complex networks, classification, high level, modularity**

I. Introduction

The term *complex network* refers to a *graph* generated by a large quantity of *nodes* (or vertices) joined by *links* (or edges), with a non-trivial topology [1]. The study of complex networks is inspired by empirical examples from the real-world, such as the internet [2], biological neural networks [3], social networks among individuals [4], food chains [5], blood distribution networks [6] and power grid distribution networks [7]. This framework has also been applied successfully to perform different types of machine learning oriented tasks, such as *clustering* [8], *classification* [9], [10] and *regression* [11]. One of the main advantages of the use of complex networks for accomplishing these tasks is the possibility of not only analyze physical features of the input data (e.g., distance or distribution), as in traditional machine learning techniques, but to also consider the pattern formation in their topological properties [9], [12]. Such type of approaches, which take into account not only physical features, but also the organizational structure of the data, are referred to as *high level* machine learning techniques.

One of the key aspects when employing a graphical approach to a classification problem lies in how the network is built from the information provided by the input dataset [13]. Usually, for this process, each data instance is mapped as a node in the network, and the edges among them are generated according to the distance they are from each other, pairwise, based on a threshold parameter. This threshold can be set either as a fixed value, thus assuming the form of a radius in the dimension space, for example, or it can also be based on a KNN algorithm, where we hence have that a node will be connected to its k nearest neighbors in the network, disregarding how far they are from each other. However, although the procedure of mapping every data instance as a node in the network is still the most adopted one, it has the drawback of, oftentimes, discarding useful information from the dataset during the mapping process. Specially when the number of features in the dataset is larger, which turns more difficult to transfer all these information to the network without any loss, since some simplifications are required for the mapping process.

In this work, we introduce a different technique for building the network, where, instead of mapping each instance of the dataset as a node, as usual, we map each of the instance's attributes as being a node. Although this procedure will overall require more memory consumption from the part of the model, it has the important advantage of allowing to preserve more information from the dataset when building the network, and hence to also make use of this extra information for improving its learning process, during the training phase. In addition, we also introduce a technique for building the network which aims to minimize the value of the threshold parameter responsible for determining the edges among the nodes, while still yielding a network with only one component per class. This technique is used in our model for the sake of turning the network more sensitive to the addition of new instances, in terms of its modularity measure, during the testing phase.

The proposed model is evaluated by applying it to benchmark artificial and real classification datasets, as well as by comparing its performance with the ones achieved by other traditional classification models on the same data. The results obtained are encouraging, with the model being ranked on second place among the 10 classifiers considered, both for the

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

artificial and real selected datasets.

Besides this introduction, this paper is organized as follows. In subsection II-A, there is an overview of the proposed model. In subsection II-B, we provide a detailed description of the model's training phase, showing how it maps the input dataset to an attributes network. In subsection II-C, we describe the model's testing phase, explaining how the probabilities for a new testing instance of belonging to each class are calculated. In section III, we list the datasets and models used for evaluating and comparing the model's performance with those obtained by other traditional classification models. In section IV, we present the obtained results, along with some discussions. At the end, in section V, we conclude this study by adding some final remarks.

II. Model Description

In this section we start, in subsection II-A, by providing an overview of the model. In subsection II-B, we have a detailed description of the model's training phase, explaining how it builds the attributes network and how it calibrates the threshold parameter. In subsection II-C, there is the description of the testing phase, explaining how the new instances are classified.

A. Model overview

In *supervised learning*, initially we have an input dataset comprising m instances and n features, in the form of $X = \{x_1, x_2, ..., x_m\}$, where each instance i consists of n dimensions, such that $x_i = (x_{i,1}, x_{i,2}, ..., x_{i,n})$, as in the following 2d array:

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & ... & x_{1,n} \\ x_{2,1} & x_{2,2} & x_{2,3} & ... & x_{2,n} \\ ... & ... & ... & ... & ... \\ x_{m,1} & x_{m,2} & x_{m,3} & ... & x_{m,n} \end{bmatrix} \quad . \tag{1}$$

The labels for each instance i are usually provided in the form of $Y = \{y_1, y_2, ..., y_m\}$, such that $y_i \in \mathcal{L} = \{l_1 ... , l_n\}$, which represent the n classes in the dataset. The objective of the *training phase* is to explore the correlations and to identify possible patterns in the dataset, in order to learn the mapping $X \xrightarrow{\Delta} Y$. To measure the level of learning, in the *testing phase*, normally we make use of a second dataset. In the absence of a second dataset, then the initial dataset can be split into two subdatasets: $X_{training}$ and $Y_{training}$, to be used for training the classifier, and X_{test} and Y_{test}, whose values of Y are suppressed so they can be used for evaluating the classifier's performance.

A network can be defined as graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes and $\mathcal{E}$ is a set of tuples representing the edges between each pair of nodes $(i, j) : i, j \in \mathcal{V}$. In the proposed model, each node in the network represents an attribute d of an instance x_i in the dataset, such that the size of G is always equal to $m \cdot n$. The connections (or edges) between nodes from different instances are created based on how far they are from each other, in the spacial dimension formed by each attribute d, according to a threshold parameter α.

The main novelty introduced by the proposed model lies in its network formation algorithm, which involves a calibration process that aims to output a network with one component for each class in the dataset, while still keeping a minimum value for the threshold parameter α. In this way, the network resulted from the training phase will be more sensitive, in terms of its modularity measure, to the insertion of new instances during the testing phase, which is expected to contribute for increasing the performance of a classification method which is solely based on this measure, as it is the case of the rationale behind the model proposed in this work. In the training phase, the attributes network is built by using the minimum possible values for the parameter α, adjusted to each dataset. In the testing phase, new instances are then inserted in the network and its label will be yielded basically by the class which results in the higher positive impact on the network's modularity measure, weighted by two other parameters, γ and ρ. Below, we provide more details about the training and testing phases of the model, along with examples of the application of its training phase on four benchmark classification datasets.

B. Description of the training phase

The training phase starts by generating the initial values of the parameter α, used for creating the edges in the network. Afterwards, it generates the attributes network G, while it calibrates the values of α in order to keep them at a minimum necessary value to obtain one component for each class in the dataset. For each class l, the parameter α_d^l yields the maximum difference between the values of attributes $x_{i,d}$ and $x_{j,d}$, with $i \neq j$, in order to connect their respective nodes by a link in the network. Its initial values are given by:

$$\alpha_d^l = \theta \sqrt{\frac{\sum_{i=1}^{m} (x_{i,d} - \bar{X}_d)^2}{m-1}} \quad , \tag{2}$$

which yields the standard deviation of the sample comprising all m values for the attribute d in $X_{training}$ multiplied by a predefined value θ, such that $\theta \in [0, 1]$. Note that those are the initial values for α, which oftentimes will change later, during the calibration of the parameter α when generating the attributes network. Therefore, as lower the value of θ, the lower will be the initial values of α and more changes are expected to occur for its values during the calibration process.

After generating the initial values for α, the next step in the training phase is to create the network from the training dataset, in which every attribute will become a node in the network and the edges between them are defined by the parameter α. We start by generating a new graph G, with $m \cdot n$ vertices, where m is the number of instances and n is the number of dimensions in $X_{training}$. Then, we create the "intra-item" edges, by taking all nodes representing an attribute $x_{i,d}$, from the same instance i, and connecting them, pairwise, such that any attribute x_{i,d_1} will be connected to the attribute x_{i,d_2}, except when $d_1 = d_2$. At this point, we will already have a network as it is shown in Figs. 1(a), 1(c), 1(e) and 1(g), with each node representing an attribute and still without any edges connecting nodes from different instances.

Afterwards, the model starts to calibrate the parameter α by, at each iteration, setting its values as the minimum distance between the components from a same class in the network and connecting then by new edges, correspondingly, until we have only one component per class, as it is shown in Figs. 1(b), 1(d), 1(f) and 1(h).

The complete process for generating the network is outlined in Algorithm 2. On lines (2:5), we create the network, add the vertices and the intra-item edges. On lines (6:28), there is the α calibration process, when the "inter-edges", i.e., the edges connecting attributes from different instances, are created, by determining the minimum values for α such that, at the end, we have one component for each class in the network.

The Algorithm 1 shows how the edges for connecting nodes from different instances in the network are created, according to the current values of the parameter α. In this case, the model will generate edges between each pair of nodes $x_{i,d}$ and $x_{j,d}$ representing a same attribute d and a same class l, where $i \neq j$, whenever the distance between them are within the range α_d^l. In this way, the total number of edges between two instances will range from 0 until the number of dimensions n in the dataset.

Algorithm 1 Inter-items edges generation

```
1:  procedure GetInterEdges(X, α, l)
2:      es ← []
3:      combs ← combinations of indexes in X^l, pairwise
4:      for i in indexes of X^l do
5:          pairs ← combs^{pair_0 = i}
6:          for p in pairs do
7:              for d = 0 to n do
8:                  dist = |X^l_{p_0,d} − X^l_{p_1,d}|
9:                  if dist ≤ α^l_d then
10:                     es ← add pair p
        return es
```

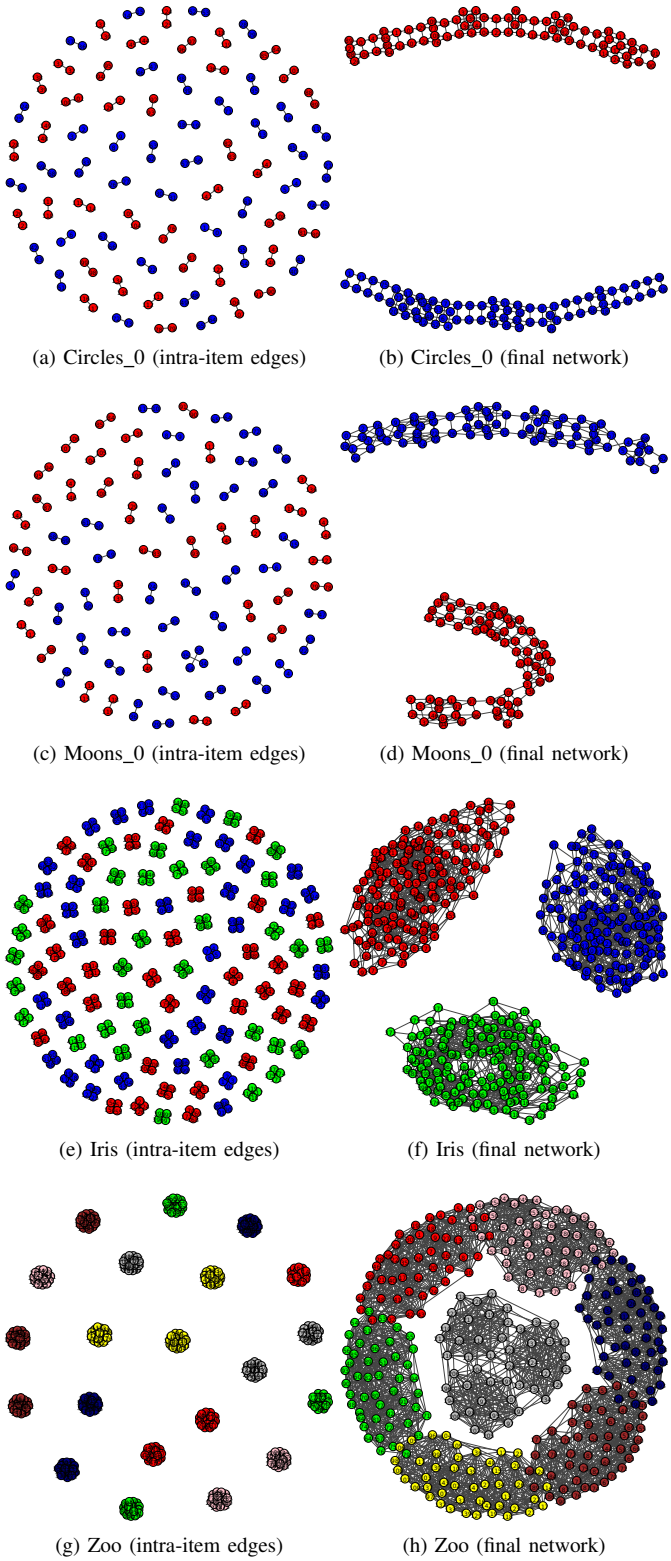
The last step in the training phase consists of generating the final values of the parameter α, to be used in the testing phase. These values are given by the arithmetic mean of the current values of α at each iteration during the calibration process. Hence, for testing purposes, the values of α are yielded by:

$$\alpha_d^l = \frac{\sum_{i=1}^{n} \alpha_{di}^l}{n} \quad , \tag{3}$$

where n is the number of changes occurred for α_d^l during its calibration process, d is one of the dimensions in the dataset and $l \in \mathcal{L}$. In this way, for testing purposes, the values of α will be somewhere between its initial values, when generally only a few edges are generated or even none of them, and its final values in the calibration process.

C. Description of the testing phase

In the testing phase, a new instance is inserted in the network G and the model then needs to assign it a label, according to its attributes values. To accomplish this task, the model starts by simulating its insertion on each of the



(a) Circles_0 (intra-item edges) (b) Circles_0 (final network)

(c) Moons_0 (intra-item edges) (d) Moons_0 (final network)

(e) Iris (intra-item edges) (f) Iris (final network)

(g) Zoo (intra-item edges) (h) Zoo (final network)

Fig. 1. Examples of networks generated by following the steps in the training phase for four datasets. (Left) network with only edges between attributes of a same instance, and (Right) final attributes network for: (a) and (b) Circles_0 dataset (2 classes and 2 features), (c) and (d) Moons_0 dataset (2 classes and 2 features), (e) and (f) Iris dataset (3 classes and 4 features), and (g) and (h) Zoo dataset (7 classes and 16 features). The number of training instances is reduced for the Zoo dataset only for the sake of visibility.

Algorithm 2 Attributes network G generation

procedure GENERATE G(X, Y)
  $vs \leftarrow m \cdot n$
  $G \leftarrow$ new graph with vs vertices
  $G \leftarrow$ add intra-item edges
  $es\_list \leftarrow []$
  while $len(G^{components}) > n$ **do**
    for $class\ in\ set(Y)$ **do**
      $es \leftarrow GetInterEdges(X, \alpha, class)$
      $es \leftarrow e$ in es if e not in es_list
      $G \leftarrow$ add edges es
      $es\_list \leftarrow$ add es
    $comps \leftarrow$ components in G
    $classes \leftarrow$ [l if $len(comps^{class=l}) > 1$]
    $C \leftarrow$ largest component in $comps^{class}$ in $classes$
    $others \leftarrow$ [other components in $comps^{class=C^{class}}$]
    $difmin \leftarrow [\infty] \cdot n$
    for $node1$ in C **do**
      $i1 \leftarrow X$ values for $node1$
      for $node2$ in $others$ **do**
        $i2 \leftarrow X$ values for $node2$
        $dif \leftarrow [0] \cdot n$
        for $d = 0$ to n **do**
          $dif_d = |i1_d - i2_d| - \alpha_d^{C^{class}}$
        if $max(dif) < max(difmin)$ **then**
          $difmin = dif$
    for $d = 0$ to n **do**
      if $difmin_d > 0$ **then**
        $\alpha_d^{C^{class}} += difmin_d$
  return G

network's components, generating edges according to the values of the parameter α, and then calculates the impacts of this insertion on the network's *modularity* measure Q, for each component, i.e., for each class. The probabilities for the new instance to belong to each class l are yielded based on the number of edges generated for each attribute d, weighted by a respective parameter γ_d, and on the impacts on the modularity Q, weighted by a respective parameter ρ^l, such that the higher is the positive impacts on Q when the instance is inserted on the component from class l, the higher will be its probability to belong to this class.

The modularity measure, broadly speaking, compares the number of connections between vertices which share a same characteristic with the expected number of connections when occurred randomly, and it is often used for detecting communities in a network. The *fast greedy* algorithm [14], for example, determines the optimal number of communities in the network by maximizing the modularity score of the graph. For the classification task proposed in this work, we take into account two other factors regarding this measure, which are: (1) How meaningful is each of these new connections (or edges) induced by the insertion of the new instance in the network and (2) The ratio between the respective number of new connections generated and the size of the network component, since, overall, larger components tend to receive more links from the new instance, which would incur in biased estimations from the classifier. Therefore, for the testing phase, we adopt two parameters, γ and ρ, for managing these two mentioned factors in the model.

The parameter γ has the role of yielding the correlations between each attribute in $X_{training}$ and the classes in $Y_{training}$. These values are used for weighting the number of edges generated between a new instance and the already existing nodes in the network, such that attributes which are more correlated to $Y_{training}$ have higher weights on the final probability scores yielded for each class. In order to determine these weights, we opt for making use of the Ridge Regression [15], with the values of γ hence assuming the values of the coefficients w returned by this linear regression model. Ridge Regression reduces the standard errors by minimizing the following loss function:

$$|Y - Xw|^2 + \lambda |w|^2 \quad , \tag{4}$$

where λ is a term to control the regularization strength. The main distinction of Ridge Regression among other linear regression models is that it enforces the coefficients w to be lower, as in the first term of (4), by introducing a constraint as the second term in (4) which penalizes large values for w. So the lower is the value of constraint λ on the features, the more the model will resemble an Ordinary Least Squares model. In our adaptation, the values of γ are yielded by:

$$\gamma_d = \frac{\sqrt{|w_d|}}{\sum_{d=1}^{n} \sqrt{|w_d|}} \quad , \tag{5}$$

where $|w_d|$ is the absolute value of the coefficient returned by the Ridge Regression model for attribute d and n is the number of dimensions in the dataset. The root square in (5) is inserted for the sake of balancing the values of γ in cases when the differences between them become too large.

The third parameter of the model ρ is responsible for balancing the impacts on the network's modularity measure, when inserting new instances during the testing phase, according to the ratio of the number of instances per class in the training dataset. This is necessary for dealing with cases of unbalanced datasets. Its values are given by:

$$\rho_l = \frac{|X_{training}|}{|X^l_{training}|} \quad , \tag{6}$$

where $l \in \mathcal{L}$ and $|X^l_{training}|$ stands for the length of all instances in $X_{training}$ from class l. The final values of ρ must also be normalized, hence assuming the form $\rho_l / \sum_l \rho_l$.

Likewise in the training phase, the new instance has its attributes mapped as n nodes in the network, where n is the number of dimensions in the dataset, and the edges among its nodes (intra-item edges) are created according to the same rule used in the training phase, with all its pairs of attributes x_{i,d_1} and x_{i,d_2} being connected, pairwise, as long as $d_1 \neq d_2$. The generation of edge $(x_{i,d}, x_{j,d})$ between the node representing

attribute d of the new instance i and any already existing node in the network j, also representing attribute d, from class l, is yielded by:

$$(x_{i,d}, x_{j,d}) = \begin{cases} 1, \text{ if } |x_{i,d} - x_{j,d}| \leq \alpha_d^l \\ 0, \text{ otherwise }, \end{cases} \tag{7}$$

where $l \in \mathcal{L}$. The total number of edges created are then averaged by each attribute d and weighted by the respective parameter γ, providing us with the indicator E, for each class l, according to:

$$E^l = \gamma_d \frac{\sum_{d=1}^{n} \epsilon_d}{n} \quad , \tag{8}$$

where ϵ_d represents the total number of edges generated for attribute d between the new instance nodes and the other nodes in the network and n is the number of dimensions in the dataset. These values are later normalized, by:

$$E^l = \frac{E^l}{\sum_{l=1}^{n} E^l} \quad , \tag{9}$$

where n is the number of classes in $\mathcal{L}$. Next, the overall impacts I^l of the new instance insertion on the network's modularity measure Q, for each class l, is calculated and has its value weighted both by E and by the parameter ρ, according to:

$$I^l = \rho^l E^l \frac{(Q^l - Q_0)}{Q_0} \quad , \text{ and} \tag{10}$$

with Q_0 being the value of the network's modularity measure Q at the end of the training phase. These values are also later normalized, providing us with what will be the probabilities $P(i^{class=l})$ of the new instance i to belong to each class l, in the form of:

$$P(i^{class=l}) = \frac{I^l}{\sum_l I^l} \quad . \tag{11}$$

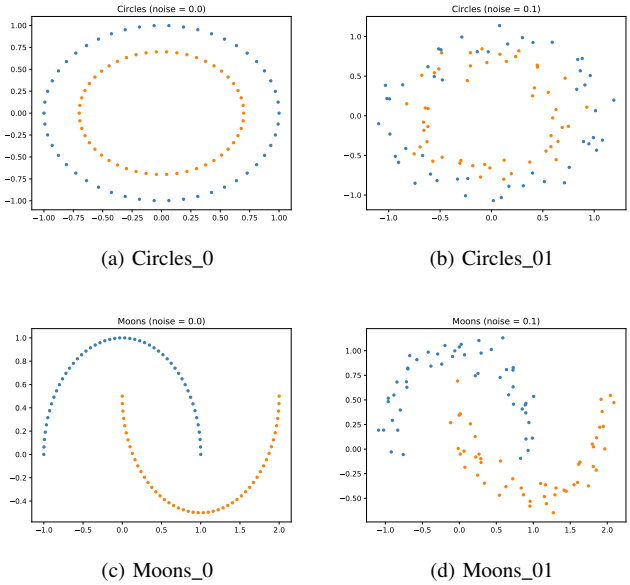
At the end, the final label l to be assigned to the new instance i will be the one among $l \in \mathcal{L}$ which maximizes $P(i^{class=l})$, being yielded by:

$$i^l = \mathcal{L}_{\arg\max_l P(i^{class=l})} \quad . \tag{12}$$

Therefore, the new instance i will belong to the class l which results in the highest positive impact on the network's modularity measure Q, when weighted by the balancing parameter ρ and also by the indicator E, which, by its turn, measures the level of "meaningfulness" of its connections in the network, so to speak, by weighting the number of edges generated per attribute by the respective correlation γ between each feature and the labels in the dataset.

III. Materials and Methods

We evaluate the efficiency of the proposed modularity-based high level model, hereafter to be mentioned as MBHL, by applying it to artificially generated data and also to well-known benchmark datasets intended for machine learning classification tests. A succinct meta-information of the selected datasets used for testing purposes is given in Table I. For a detailed description of the real datasets, one can refer to [16]. Examples of the artificial datasets generated for the tests are provided in Fig. 2. For splitting each dataset into 2 subdatasets, for training and testing purposes, we made use of a function which shuffles the data, through a random seed value, and returns a train-test split with 75% and 25% the size of the inputs, respectively. As preprocessing, all real datasets have their features treated through a quantile transformation, such that their values are adjusted to follow a uniform distribution, ranging from 0 to 1.



(a) Circles_0 (b) Circles_01

(c) Moons_0 (d) Moons_01

Fig. 2. Artificial datasets generated for the evaluation and comparison of the model. Above: (a) two concentric circles without noise and (b) two concentric circles with a noise of 0.1. Below: (c) two moons without noise and (d) two moons with a noise of 0.1.

TABLE I
META INFORMATION OF THE CLASSIFICATION DATASETS USED FOR EVALUATING AND COMPARING THE MBHL MODEL

		N^o of Samples	N^o of Features	N^o of Classes
Artificial	Circles_0	100	2	2
	Circles_01	100	2	2
	Moons_0	100	2	2
	Moons_01	100	2	2
Real	Bankruptcy	250	6	2
	Haberman	306	3	2
	Hayes-roth	132	5	3
	Iris	150	4	3
	Wine	178	13	3
	Zoo	101	16	7

For the sake of comparison, the following traditional classification models are applied on the same datasets listed in Table I: Decision Tree [17], Logistic Regression [18], Multilayer Perceptron [19], Support Vector Machines with an RBF kernel [20] and Naive Bayes [21]. We also apply the following ensemble methods: Bagging of Decision Tree and Bagging of MLP [22], Random Forest [23] and AdaBoost [24]. All traditional models are implemented through [25]

and we keep their respective default parameters values, in all tests performed. As for the proposed MBHL model, we set the parameters $\theta = 0.1$ for tests with artificial datasets and $\theta = 0.5$ for tests with real datasets. The value of λ, for the Ridge Regression, was set to 1.0. For the tests with artificial datasets, the model generates edges among nodes from different instances only when all distances between them are within the respective range yielded by α_d. Each dataset is processed 50 times by all models, each time having their data items shuffled by using a different randomly generated seed. The final accuracy scores are the averaged ones achieved by each model, on each dataset.

IV. Results and Discussion

In this section, we present the obtained results when applying the proposed MBHL model to artificial and real benchmark classification datasets, along with a comparison of its performance with the ones achieved by traditional classification models on the same data.

The results obtained from the application of the proposed MBHL model, along with other traditional classification models, both on real and artificial datasets, are summarized in Table II. The Average Rank, in the last row, indicates the averaged rank position achieved by each model considering all datasets, according to their respective rank achieved on each of them, in terms of mean accuracy values.

Regarding the results obtained on artificial datasets, although the MBHL model was not ranked so well for the Circles_01 dataset (two concentric circles with 0.1 noise), being ranked on fifth place, it was still able to achieve the second place in the average rank. This is because its performance on the other three datasets was very stable, having achieved second place in all of them. Note that, for this database, the RBF SVM model is ranked as first, in all datasets, and the MBHL model is ranked right after it in the average rank, followed by the AdaBoost model, on third place.

Both Circles and Moons datasets can be challenging to classify due to their non-linearity property, specially when noise is inserted in the problem. Classifiers which are based mainly on the linear distance among instances tend to perform poorer in this type of scenario, since testing instances from different labels get more mixed, and the classes oftentimes overlap each other in the decision space. In this sense, the relative good performance of MBHL on these type of datasets, finishing on second place overall, indicates that the model is able to correctly detect the topological patterns formation, for the selected datasets, and to adjust its inference process according to these identified patterns.

If we look at Figs. 1(b) and 1(d), which show the attributes networks resulted from the training phase for Circles_0 and Moons_0 datasets, respectively, we can note that only the nodes representing instances which are immediately adjacent to each other in these datasets become connected in the training phase. This happens due to two factors: (1) The form with which the parameter α is calibrated when building the network, by keeping it at a minimum value in order to generate only the enough number of edges for allowing the connection between all components from a same class in the network, and (2) The rule where the model will connect the nodes of different data instances only if all distances between them are within their respective thresholds yielded by the parameter α, for all dimensions considered. Hence, the capacity of the model to identify the topological patterns in a dataset comes from these two factors combined. In this way, the network is more sensitive to the disturbances in its topology provoked by the insertion of new instances during the testing phase. Also, nodes representing instances which are close from each other in one of the dimensions, but are far from each other in other dimensions, do not get connected, since all features of the dataset are considered when generating the edges between different instances.

As for the obtained results when applying the proposed MBHL model on real datasets, we can note, in Table II, that it achieves its best relative performance on Zoo dataset, being ranked as second best, right after the Logistic Regression model and followed by Random Forest model. Likewise it happened on the tests with artificial datasets, although its relative performance do not really stand out from the other classifiers, when it comes to another real datasets, it was still able to achieve second place in the average rank since it was overall more stable than the other models, in terms of relative performance, having always being ranked as 4th or 5th on the other datasets.

The good relative performance of the MBHL model on the Zoo dataset – which has a total of 16 features, with most of them being binary ones – is a sign that the parameter γ is properly fulfilling its role, of measuring the "meaningfulness" of each attribute in the classification task. For this specific dataset, for example, the 16 values for this parameter were: (0.07, 0.09, 0.07, 0.1 , 0.03, 0.06, 0.05, 0.03, 0.11, 0.08, 0.03, 0.04, 0.07, 0.11, 0, 0.07). Therefore, as we can note, none of the weights is higher than 0.11, and only one of them is set as 0, which means that the model identified this attribute as not meaningful for the classification task. It is also worth noting that the Zoo dataset has 7 classes and only 101 samples, which indicates that the model can also learn well even when the number of data instances per class is limited. As for the results obtained on the Haberman dataset, we would like to point out that, after 50 random train-test data splits, half of the models (5 out of 10) still achieved a mean accuracy of less or equal to 50% on it. Considering that this dataset has only 2 classes, then it means that this classification problem is a challenging one, and that the mean accuracy achieved by the MBHL model on it, of 59.4%, is quite satisfactory.

V. Final Remarks

In this work, we introduce a high level classification model which maps each attribute of the dataset as a node of a network, and the labels assigned to testing instances are based mainly on the modularity measure. For its evaluation, the model was applied on both artificial and real benchmark datasets, and had its performance compared to the

TABLE II

RESULTS: MEAN ACCURACY RATES FOR EACH DATASET OBTAINED BY THE FOLLOWING MODELS, IN THAT ORDER: MBHL, ADABOOST, BAGGING OF DECISION TREE, BAGGING OF MLP, DECISION TREE, LOGISTIC REGRESSION, MLP, NAIVE-BAYES, RANDOM FOREST AND SVM. THE VALUES BETWEEN PARENTHESIS INDICATE THE RANK ACHIEVED BY EACH MODEL ON EACH DATASET.

		MBHL	Ada	BagDT	BagMLP	DT	LR	MLP	N-B	RF	SVM
Artificial	Circles_0	0.985 (2)	0.983 (3)	0.976 (5)	0.725 (9)	0.982 (4)	0.388 (10)	0.797 (7)	0.784 (8)	0.969 (6)	1.0 (1)
	Circles_01	0.796 (5)	0.820 (3)	0.818 (4)	0.696 (9)	0.795 (6)	0.396 (10)	0.750 (8)	0.762 (7)	0.831 (2)	0.896 (1)
	Moons_0	0.988 (2)	0.979 (3)	0.922 (6)	0.857 (8)	0.933 (5)	0.844 (10)	0.855 (9)	0.864 (7)	0.968 (4)	0.998 (1)
	Moons_01	0.973 (2)	0.955 (3)	0.912 (5)	0.852 (9)	0.909 (6)	0.845 (10)	0.860 (7)	0.859 (8)	0.951 (4)	0.976 (1)
	Average Rank	2nd	3rd	5th	9th	6th	10th	8th	7th	4th	1st
Real	Bankruptcy	0.993 (4)	0.996 (2)	0.995 (3)	0.961 (8)	0.995 (3)	0.962 (7)	0.963 (6)	0.957 (9)	0.997 (1)	0.986 (5)
	Haberman	0.594 (4)	0.465 (9)	0.496 (7)	0.651 (2)	0.489 (8)	0.700 (1)	0.649 (3)	0.534 (5)	0.500 (6)	0.496 (7)
	Hayes-roth	0.651 (4)	0.592 (7)	0.712 (2)	0.553 (9)	0.690 (3)	0.513 (10)	0.569 (8)	0.633 (5)	0.721 (1)	0.613 (6)
	Iris	0.938 (5)	0.930 (7)	0.950 (2)	0.918 (8)	0.930 (6)	0.913 (10)	0.918 (9)	0.945 (3)	0.940 (4)	0.965 (1)
	Wine	0.961 (5)	0.700 (10)	0.927 (8)	0.932 (7)	0.894 (9)	0.964 (3)	0.940 (6)	0.961 (4)	0.971 (2)	0.977 (1)
	Zoo	0.927 (2)	0.801 (9)	0.904 (7)	0.925 (4)	0.860 (8)	0.928 (1)	0.925 (5)	0.925 (5)	0.926 (3)	0.918 (6)
	Average Rank	2nd	9th	4th	8th	7th	6th	7th	5th	1st	3rd

ones achieved by other traditional classification models. The obtained results on artificial datasets indicate that the model is able to correctly detect topological patterns in the data, including those with non-linearity properties, and to adjust its inference process accordingly. The results obtained from its application on real datasets were also encouraging, with the model being able to achieve competitive mean accuracy rates when compared to traditional classifications models.

As future research, we plan to work on the improvement of some of the model's current features, such as: (1) Possibly eliminate or automatize the parameter θ, so that the model may be able to learn and to infer the information regarding this parameter directly from the characteristics of each dataset, during the training phase, (2) Although our current choice for adopting the Ridge Regression for the sake of generating the values of the parameter γ has demonstrated, based on the obtained results, to be quite fair so far, we still consider that other forms of generating these values should definitely be explored in the future, and (3) Optimize the model's memory consumption and its processing time, specially in the α calibration process since, at the way it is now, this process can take too long as the number of attributes in the dataset increases, hence such optimizations would allow testing the model with larger datasets, such as images, for example.

ACKNOWLEDGMENTS

This work is supported in part by the São Paulo State Research Foundation (FAPESP) under grant numbers 2015/50122-0 and 2013/07375-0, the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and the Brazilian National Council for Scientific and Technological Development (CNPq) under grant number 303199/2019-9.

REFERENCES

[1] R. Albert and A. L. Barabási, "Statistical mechanics of complex networks." *Reviews of Modern Physics*, vol. 74, pp. 47–97, 2002.

[2] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology." *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4, 1999.

[3] O. Sporns, "Network analysis, complexity, and brain function." *Complexity*, vol. 8, no. 1, pp. 56–60, 2002.

[4] P. J. Carrington, J. Scott, and S. Wasserman, *Models and methods in social network analysis.* Cambridge: Cambridge University Press, 2006.

[5] J. M. Montoya and R. V. Sol, "Small world patterns in food webs." *Journal of Theoretical Biology*, vol. 214, no. 3, pp. 405–412, 2002.

[6] G. B. West, J. H. Brown, and B. J. Enquist, "A general model for the structure, and allometry of plant vascular systems." *Nature*, vol. 400, pp. 125–126, 2009.

[7] R. Albert, I. Albert, and G. L. Nakarado, "Structural vulnerability of the north american power grid." *Physical Review*, vol. 69, no. 2, p. 025103, 2004.

[8] T. C. Silva and L. Zhao, "Stochastic competitive learning in complex networks," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 23, no. 3, pp. 385–398, 2012.

[9] ——, "Network-based high level data classification." *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 23, no. 6, pp. 954–970, 2012.

[10] T. Colliri and L. Zhao, "Analyzing the bills-voting dynamics and predicting corruption-convictions among Brazilian congressmen through temporal networks," *Scientific Reports*, vol. 9, no. 1, pp. 1–11, 2019.

[11] X. Gao, H. An, W. Fang, X. Huang, H. Li, W. Zhong, and Y. Ding, "Transmission of linear regression patterns between time series: From relationship in time series to complex networks," *Physical Review E*, vol. 90, no. 1, p. 012818, 2014.

[12] T. Colliri, D. Ji, H. Pan, and L. Zhao, "A network-based high level data classification technique," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[13] L. Berton, A. de Andrade Lopes, and D. A. Vega-Oliveros, "A comparison of graph construction methods for semi-supervised learning," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, pp. 1–8.

[14] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Physical Review E*, vol. 70, no. 6, p. 066111, 2004.

[15] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[16] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[17] S. R. Safavin and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, no. 3, pp. 660–674, 1991.

[18] A. Gelman and J. Hill, *Data analysis using regression and multilevel-hierarchical models*. Cambridge University Press New York, NY, USA, 2007, vol. 1.

[19] G. E. Hinton, "Connectionist learning procedures," *Artificial intelligence*, vol. 40, no. 1-3, pp. 185–234, 1989.

[20] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer, 2000.

[21] I. Rish, "An empirical study of the naive bayes classifier," *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22, 2001, IBM New York.

[22] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[23] ——, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[24] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *European Conference on Computational Learning Theory*. Springer, 1995, pp. 23–37.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.