A Network-Based High-Level Data Classification Algorithm Using Betweenness Centrality

Esteban Wilfredo Vilca Zuñiga[1], **Liang Zhao**[1]

[1] Dept. of Computing and Mathematics
FFCLRP-USP
Ribeirão Preto, Brasil

evilcazu@usp.br, zhao@usp.br

Abstract. *Data classification is a major machine learning paradigm, which has been widely applied to solve a large number of real-world problems. Traditional data classification techniques consider only physical features (e.g., distance, similarity, or distribution) of the input data. For this reason, those are called low-level classification. On the other hand, the human (animal) brain performs both low and high orders of learning and it has a facility in identifying patterns according to the semantic meaning of the input data. Data classification that considers not only physical attributes but also the pattern formation is referred to as high-level classification. Several high-level classification techniques have been developed, which make use of complex networks to characterize data patterns and have obtained promising results. In this paper, we propose a pure network-based high-level classification technique that uses the betweenness centrality measure. We test this model in nine different real datasets and compare it with other nine traditional and well-known classification models. The results show us a competent classification performance.*

1. Introduction

Machine learning can be defined as a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict the future data, or perform other kinds of decision making under uncertainty [Murphy 2013].

Usually, machine learning is divided into three main paradigms: *supervised learning*, *unsupervised learning*, and *semi-supervised learning* [Géron 2017]. Supervised learning uses tagged data to detect patterns and predict future cases. According to the kind of labels, the prediction is called *classification* for categorical labels and *regression* for numerical labels. The unsupervised machine learning algorithms explore the data to search for possible structures that can tag the data. For example, on social media, the users don't provide necessarily some special information like political preferences. However, using the collected data an unsupervised algorithm could detect this [Géron 2017]. Semi-supervised learning is a combination of supervised and unsupervised learning. Usually, the quantity of labeled data is low because tagging is expensive. So, semi-supervised algorithms use some labeled data to predict the untagged data [Patel 2019].

Data classification is one of the most important topics in supervised learning. It aims at generating a map from the input data to the corresponding desired output, for a

given training set. The constructed map, called a classifier, is used to predict new input instances.

Many algorithms use only physical features (e.g., distance, similarity, or distribution) for classification. These are called *low-level* classification algorithms. Such algorithms can get good classification results if the training and testing data are well-behaved, for example, the data satisfies a normal distribution. But, these techniques have a problem to classify data with complex structures. On the other hand, a high-level algorithm uses data interaction as a system for classification, exploiting the structure of the data to capture patterns. In this way, it can perform classification according to pattern formation of the data like cycles, high links density, assortativity, network communication (*betweenness*), and so on instead of just measuring physical features like euclidean distance.

In order to capture the structure and properties of the data, we propose to work with complex networks, which are defined as large scale graphs with nontrivial connection patterns [Albert and Barabási 2002]. Many network measures have been developed and each one of them characterizes network structure from a particular viewpoint. In the category of degree-related measures, we have the *density* that represents how strong the nodes connections are [Christiano Silva and Zhao 2016] and *assortativity* degree that represents the attraction of nodes with a similar degree (degree correlation) [Newman 2003]. In the category of *centrality measures*, we have *betweenness centrality* that measures the node importance for communication on the network [Freeman 1977], *closeness vitality* that measures the impact of a network communication if a node is removed [Christiano Silva and Zhao 2016], and so on.

Several high-level algorithms have been proposed to use network measures to make a classification. *Impact measure* approach tries to reduce the variation of a measure once a new node is inserted into a network [Colliri et al. 2018], *link prediction* approach uses a meta class node to represent each label and the classification is performed using link predictions techniques [Fadaee and Haeri 2019], and *importance measure* exploits the page-rank algorithm for classification [Carneiro and Zhao 2018].

The technique proposed in this work, captures the structure of data using just one metric $betweenness\ centrality$. This measure captures the node importance for the graph communication. Nodes that have low $betweenness$ tend to be on the periphery on the contrary the nodes tend to be focal points [Christiano Silva and Zhao 2016]. Instead of focusing on the node insertion impact or preservation of the structure measure using many network measures. We focus on the structure generated once a new node is inserted and identify if this inserted node presents similar features to the others in the new network. Also, unlike other methodologies that require a classical algorithm like SVM (Support Vector Machine) to complete the high-level classification [Silva and Zhao 2012], our methodology uses pure network measures to classify. This approach shows good performance, avoids the double calculations of *impact measure* method, reduces the number of properties to be used, and do not require to be combined with other classical techniques.

2. Model Description

In this section, we describe the working mechanism of our model. Firstly, we give an overview of the training and classification model phase. Then we provide details about each step of the algorithm. Finally, we describe how we use the betweenness measure on

the model.

2.1. Overview of the Model

Each complex network consists of a set of nodes or vertices $\mathcal{V}$ and a set of links or edges $\mathcal{E}$ between each pair of nodes. The input data $\mathcal{D}$ of N elements for $supervised\ learning$ contains two parts: the attributes $\mathcal{X}$ and the labels $\mathcal{Y}$.

In $supervised\ learning$, the dataset $\mathcal{D} = \{(X_1, y_1), ..., (X_n, y_n)\}$ where $X_i = (x_1, ..., x_d)$ represents the d attributes, and y_i represents the label of the instance X_i. The values of $y_i \in \mathcal{L} = \{l_1, ..., l_c\}$ where $\mathcal{L}$ is the possible labels of the instance. The goal of $supervised\ learning$ is to predict the y_i values using the instances X_i. This could be considered as function approximation $f(X_i) \approx y_i$ where the function f is our algorithm. To evaluate the model, it is required to split the data in training and testing datasets. The $X_{training}$ dataset will be used to build our model and the $X_{testing}$ dataset will be used for evaluation.

In the training phase, we will build complex networks using the training dataset. The instances in the dataset will be the nodes and the links will represent the similarity between these nodes. Therefore, we will have $\mathcal{D} \mapsto \mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, where $\mathcal{V} = \{1, ..., N\}$ is the set of nodes and $\mathcal{E}$ is the set of links in the complex network $\mathcal{G}$. The links could be created using kNN and $\epsilon - radius$ or personalized relation metrics like friendship on social data, flight routes, or city connections.

The network $\mathcal{G}$ will be built using $X_{training}$ to produce the nodes $\mathcal{V}$ and kNN and $e - radius$ as relation metric for links $\mathcal{E}$. Then, we remove the links between nodes with different labels y_i. Following this strategy, we will have one network component $\mathcal{G}^i$ for each label in $\mathcal{L}$.

In the testing phase, we insert a node from $X_{testing}$ into each component $\mathcal{G}^i$ following the same kNN and $e - radius$ rules of training phase. Then, we calculate the $betweenness\ measure$ of this node in each $\mathcal{G}^i$. This measure is compared to the others from each network component $\mathcal{G}^i$. So, the differences are saved in a new list for each $\mathcal{G}^i$.

Finally, we get the average of the b lowest values for each list and we classify the new node to the $\mathcal{G}^i$ with the lowest average. Then, we remove this node from the other components. In the case that the average differences of two or more lists are equal, we use the number of links connected to this new node in each component as a second difference measure.

2.2. Network-Based High Level Classification Algorithm Using Betweenness Centrality (NBHL-BC)

The proposed high-level classification algorithm, which will be referred as NBHL-BC, has four parameters k, e, b, and α. Where k is the number of neighbors used in the kNN, e is the percentile into $kNN_{distances}$ used to calculate ϵ, b is the number of nodes with similar $betweenness$ used for classification,and α is the weight to balance between links and $betweenness\ centrality$.

During the training phase, we need to build the network using the $X_{training}$ and $Y_{training}$ where $X_{training} \mapsto \mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$. Each node in $\mathcal{V}$ is related with one instance in $X_{training}$ and each link in $\mathcal{E}$ is defined following these two techniques:

$$\mathcal{N}(X_i) = \begin{cases} \epsilon\text{-}radius(X_i, y_i), & \text{if } |\epsilon\text{-}radius(X_i, y_i)| > k \\ kNN(X_i, y_i), & \text{otherwise} \end{cases} \quad (1)$$

Where (X_i, y_i) represents a pair of data instance X_i and its corresponding label y_i. For each instance X_i, $\mathcal{N}(X_i)$ is the set of nodes to be connected to it, its neighborhood. $\epsilon\text{-}radius(X_i, y_i)$ returns the set of nodes $\{X_j, j \in \mathcal{V} : distance(X_i, X_j) < \epsilon \wedge y_i = y_j\}$ i.e. the set of nodes X_j whose similarity with X_i is beyond a predefined value ϵ and have the same class label y_i. Here, $distance$ is a similarity function like euclidean distance. $kNN(X_i, y_i)$ returns the set containing the k nearest neighbors of X_i. The value ϵ is the percentile e of the $kNN_{distances}$ in the sub graph of y_i. Note that the ϵ-radius criteria is used for dense regions ($|\epsilon - radius(Xi)| > k$), while the kNN is employed for sparse regions. With this mechanism, it is expected that each label will have an independent sub graph $\mathcal{G}^c$ [Silva and Zhao 2012] [Silva and Zhao 2015] [Colliri et al. 2018].

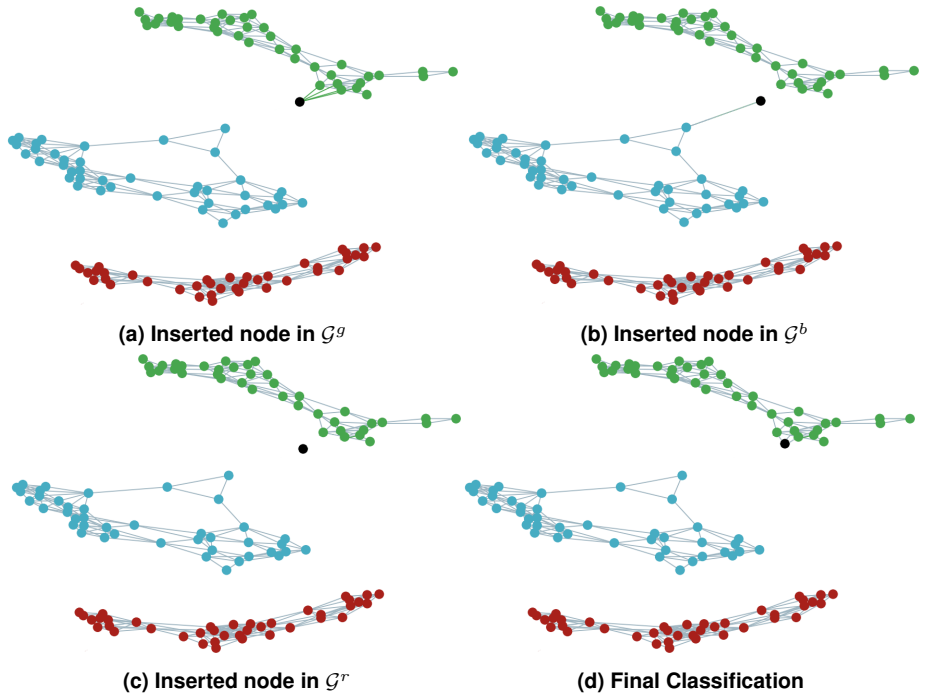


Figure 1: Classification of a new instance (dark node) into the iris dataset graph $\mathcal{G}$ using with $k = 5$ $e = 0.5$ $b = 5$ $\alpha = 1.0$

In Figure 1d, we can see the graph $\mathcal{G} = \{\mathcal{G}^r, \mathcal{G}^g, \mathcal{G}^b\}$ where r, g, b represent the sub graphs with nodes red, green and blue. On the testing phase, we insert each $X_{testing}$ instance (dark node) to each component $\mathcal{G}^i$ following the same rule on equation (1), and assuming that the node will be inserted in each sub graph.

For example, in Figure 1, there are three network components $\mathcal{G}^i$ and the node to be tested is inserted to each one. In Figures 1a 1b 1c, the node uses its k nearest neighbors with the same label. In this case with $k = 5$, there are 4 nodes in $\mathcal{G}^g$, 1 in $\mathcal{G}^b$, and 0 in $\mathcal{G}^r$. The $\epsilon - radius$ with the $e = 0.5$ (median of $kNN_{distances}$) is less than 5, because the current inserted node presents a sparse behavior; for this reason, we will use

just kNN.Moreover, due to the condition of the same label, the algorithm will produce one sub graphs for each possible label.

Now we calculate the $betweenness\ centrality$ for the node in each component when the new node is inserted. Following this rule, the inserted node will have different values for each component.

The $betweenness\ centrality$ is a mixed measure (global and local) that captures how much a given node is in the shortest paths of others nodes [Christiano Silva and Zhao 2016]. This measure captures the influence of a node in the communication of the network [Needham and Hodler 2019]. We capture not only the characteristics of a node but also the behavior of their neighborhood. So, we have a metric that provides local and global network characteristics. This metrics is defined in the equation 2.

$$B(i) = \sum_{s \neq i \in \mathcal{V}} \sum_{t \neq i \in \mathcal{V}} \frac{\eta_{st}^{i}}{\eta_{st}} \tag{2}$$

where η_{st}^{i} is 1 when the node i is part of the geodesic path from s to t and 0 otherwise. η_{st} is the total number of shortest paths between s and t.

Then, we calculate the difference of this measure between the inserted node and the other nodes in each component $\mathcal{G}^i$. In the algorithm 2 on line 14, we can show this step. In the algorithm 1, we can appreciate how an inserted node will present a different $betweenness\ centrality$ for each sub graph.

Algorithm 1 Node Insertion

1: **function** NODEINSERTION($\mathcal{G}, instance, index, k, e$)
2: $\langle \mathcal{V}, \mathcal{E} \rangle \leftarrow \mathcal{G}$ ▷ index is the number of nodes in the graph +1
3: $\mathcal{V} \leftarrow \mathcal{V} \cup \{index\}$
4: $edges \leftarrow \{\}$
5: **for** $X_i \in X_{training}$ **do**
6: **if** $X_i \in \mathcal{N}(instance, k, e)$ **then**
7: $edges \leftarrow edges \cup (i, index)$
8: **end if**
9: **end for**
10: $\mathcal{E} \leftarrow \mathcal{E} \cup \{edges\}$
11: $\mathcal{G} \leftarrow \langle \mathcal{V}, \mathcal{E} \rangle$
12: **return** $\mathcal{G}$
13: **end function**

These values will be inserted into an independent list for each component $\mathcal{G}^i$. We will calculate the average of the b lower values on each list. In the 2 on line 19, we can appreciate how we get just the b lower elements on NB previously sorted on line 16. The results are stored on the array $\mathcal{W} = \{w_1, ..., w_c\}$ where each w_i represents the average difference of the b nearest betweenness node values on the sub graph $\mathcal{G}^i$. This process is represented in the algorithm 2 on line 27 and 28.

$$\mathcal{W}^n = \frac{1 - \mathcal{W}}{\sum_{w_i \in \mathcal{W}} 1 - w_i} \tag{3}$$

Where $\mathcal{W}^n$ is the normalized version of $\mathcal{W}$. In order to avoid conflicts of probabilities with the same value $w_i \in \mathcal{W}^n$, we calculate the number of links of the inserted node with respect to each sub graph $\mathcal{G}^i$ on the array $\mathcal{T}$. Then, we follow a similar process of equation 3 for $\mathcal{T}$ normalization. This process is represented in the algorithm 2 on line 29.

$$\mathcal{T}^n = \frac{\mathcal{T}}{\sum_{t_i \in \mathcal{T}} t_i} \tag{4}$$

Finally, once we normalize these values, we calculate the sum of $\mathcal{T}^n, \mathcal{W}^n$ and made a final normalization.

$$\mathcal{H} = \frac{(\alpha)\mathcal{W}^n + (1-\alpha)\mathcal{T}^n}{\sum_{t_i \in \mathcal{T}^n, w_i \in \mathcal{W}^n} (\alpha) w_i + (1-\alpha) t_i} \tag{5}$$

where $h_i \in \mathcal{H}$ represents the probability of a node i to be inserted in the sub graph $\mathcal{G}^i$, and α controls the weights between structural information and number of links. If $\alpha = 1.0$, we just capture information using $betweenness\ centrality$, and if $\alpha = 0.0$, we just capture information about number of links. The fully algorithm is described in algorithm 2.

3. Performance Tests on Toy Datasets

In this section, we present the classification performance of our algorithm in toy datasets and compare the results with other algorithms using python as programming language and Scikit-learn library for algorithms [Pedregosa et al. 2011]. Specifically, we test our algorithm against Multi Layer Perceptron (MLP) [Riedmiller and Braun 1993], Decision Tree C4.5 (DT) [Shafer et al. 2000], and Random Forest (RF) [Breiman 2001]. The algorithms are tested using cross validation 10-folds, executed 10 times, and we use a grid search to select the hyper parameters that give the best accuracy for all the algorithms.

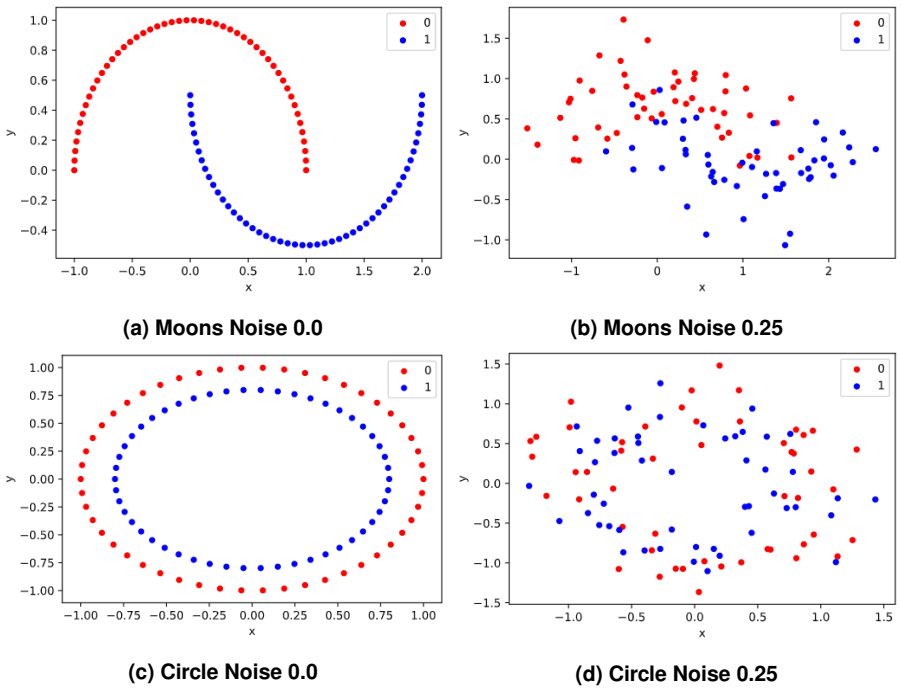
The toy datasets are Moons and Circle with 0.0 and 0.25 of Gaussian standard deviation noise added to the data 2. The NBHL-BC parameter values are shown in table 1, and the classification accuracy results are shown in table 2. These datasets were used because present clear data patterns where traditional algorithms reduce their effectiveness. In the case of Decision tree, we use gini index as quality measure without pruning method. In the case of Random Forest, we use gini index as split criterion and 100 trees. In the case of MLP, we use 2 hidden layer with 10 nodes and 100 interactions for dataset without noise and 500 interactions with noise.

We use in the first group $\alpha = 1.0$ because we want to evaluate just the structural methodology using $betweenness\ centrality$. In the second group, we combine both strategies with $\alpha = 0.5$ and we got a small improvement on the dataset Circle 0.25. In the last group, we use just the number of links and we got similar results but there is a reduction of the accuracy in Moons 0.25. In some cases, we need to remove the property of $\epsilon - radius$ using $e = 0.0$ and increase the quantity of k neighbors in Moons with 0.25 noise. The b similar $betweenness\ centrality$ nodes were kept in all the tests because other values reduce accuracy.

In this simulations, our algorithm presents the best results in all the datasets. Specially in the the circle dataset with noise 0.25, which is the most difficult case, our algorithm presents better classification accuracy than other techniques under comparison.

Algorithm 2 Classification Algorithm

function CLASSIFICATION($\mathcal{G}$,$instance, k, e, b, \alpha$)
  $index \leftarrow n + 1$ ▷ n is the number of nodes in $\mathcal{G}$
  $\mathcal{G} \leftarrow$NodeInsertion($\mathcal{G}, instance, index, k, e$)
  $\mathcal{W} \leftarrow \{\}$
  $\mathcal{T} \leftarrow \{\}$
  for $\mathcal{G}^i \in \mathcal{G}$ **do** ▷ Where each $\mathcal{G}^i$ is a subgraph
    $NB \leftarrow \{\}$ ▷ NB is a list of node betweenness differences
    $\langle \mathcal{V}^i, E^i \rangle \leftarrow \mathcal{G}^i$
    $Links \leftarrow 0$
    for $j \in \mathcal{V}^i$ **do**
      if $j \in \mathcal{N}(index, k, e)$ **then**
        $Links \leftarrow Links + 1$
      end if
      $NB \leftarrow NB \cup \{B(index) - B(j)\}$ ▷ B is betweenness centrality
    end for
    Sort(NB) ▷ NB has the differences between the nodes in $\mathcal{G}^i$ and the new node
    $Total \leftarrow 0$
    $count \leftarrow 0$
    while $count < b$ **do**
      $Total \leftarrow Total + NB[count]$
      $count \leftarrow count + 1$
    end while
    $Total \leftarrow \frac{Total}{b}$
    $\mathcal{W} \leftarrow \mathcal{W} \cup Total$
    $\mathcal{T} \leftarrow \mathcal{T} \cup Links$
  end for
  $\mathcal{W}^n \leftarrow 1 - \mathcal{W}$
  $\mathcal{W}^n \leftarrow \frac{\mathcal{W}^n}{sum(\mathcal{W}^n)}$
  $\mathcal{T}^n \leftarrow \frac{\mathcal{T}}{sum(\mathcal{T})}$
  $\mathcal{H} \leftarrow (\alpha)\mathcal{W} + (1 - \alpha)\mathcal{T}$
  $\mathcal{H} \leftarrow \frac{\mathcal{H}}{sum(\mathcal{H})}$
  return MaxIndexValue($\mathcal{H}$) ▷ $\mathcal{H}$ has each class probability
end function



(a) Moons Noise 0.0

(b) Moons Noise 0.25

(c) Circle Noise 0.0

(d) Circle Noise 0.25

Figure 2: Synthetic Datasets for Testing

Dataset	k	e	b	α	accuracy
Moons 0.00	5	0.5	5	1.0	**100.0**
Moons 0.25	8	0.0	10	1.0	**97.0**
Circle 0.00	1	0.5	1	1.0	**100.0**
Circle 0.25	5	0.5	1	1.0	64.0
Moons 0.00	5	0.5	5	0.5	**100.0**
Moons 0.25	9	0.0	10	0.5	96.0
Circle 0.00	1	0.0	1	0.5	**100.0**
Circle 0.25	5	0.5	1	0.5	**65.0**
Moons 0.00	5	0.5	5	0.0	**100.0**
Moons 0.25	9	0.0	10	0.0	96.0
Circle 0.00	1	0.0	1	0.0	**100.0**
Circle 0.25	5	0.5	1	0.0	64.0

Table 1: Parameter values used by our algorithm (NBHL-BC) in Toy Datasets

	MLP	DT	RF	NBHL-BC
Moons 0.00	94.0	95.0	98.0	**100.0**
Moons 0.25	84.0	85.0	91.0	**97.0**
Circle 0.00	90.0	92.0	91.0	**100.0**
Circle 0.25	62.0	56.0	56.0	**65.0**

Table 2: Classification accuracy of the NBHL-BC compared to Multi Layer Perceptron (MLP), Decision Tree (DT), and Random Forest (RF) in Toy Datasets

4. Experimental Results on Real Datasets

In this section, we are going to present the results of the NBHL-BC technique on UCI classification datasets [Dua and Graff 2017] . Also, we will compare our results with

other algorithms . We tested our algorithm against Multi Layer Perceptron (MLP) [Riedmiller and Braun 1993], Decision Tree C4.5 (DT) [Shafer et al. 2000], Random Forest (RF) [Breiman 2001], and the Network Base High Level Technique (NBHL) [Colliri et al. 2018].

The algorithms are tested splitting each dataset in two sub data sets, for training and testing with a proportion of 75% and 25% respectively following an stratified sampling using python as programming language and Scikit-learn library for algorithms [Pedregosa et al. 2011].

The datasets used are shown in table 3 with the number of instances, attributes and classes. These datasets were selected because the previous high-level algorithm used them. The NBHL-BC parameter values are given in table 4, and classification accuracy results are presented in table 5.

Dataset	Instances	Attributes	Classes
Glass	214	9	6
Iris	150	4	3
Pima	768	8	2
Teaching	151	5	3
Wine	178	13	3
Yeast	1484	8	10
Zoo	101	16	7

Table 3: Information about the UCI classification dataset used on these project

Dataset	k	e	b	α
Glass	1	0.0	1	1.0
Iris	7	0.0	3	1.0
Pima	8	0.0	4	1.0
Teaching	5	0.0	5	1.0
Wine	12	0.0	5	1.0
Yeast	14	0.0	3	0.5
Zoo	1	0.0	1	1.0

Table 4: Parameter values used by our algorithm (NBHL-BC) in UCI datasets

Our algorithm presents a good performance in all the datasets compared to other algorithms. In four cases, our algorithm presents the best results. Just in case of Iris dataset, another high level classification algorithm NBHL is better than the proposed one.

Moreover, the α parameter that regulates the weight between the $betweenness$ measure and number of links in 6 of the 7 datasets is 1.0 that means that the algorithm just use the $betweenness$. In the dataset Yeast, it was required an $\alpha = 0.5$ that means that give same importance between $betweenness$ and number of links. In table 6, we tested UCI Wine dataset [Dua and Graff 2017] using 10-fold cross validation with different values for α. The accuracy with only links number $\alpha = 0.0$ is quite lower than $\alpha = 1.0$, and the best result is mixing both techniques with $\alpha = 0.4$. The b parameter that evaluates the number of nodes with the lower $betweenness\ centrality$ difference with respect to the inserted node were kept constant.

	MLP	DT	RF	NBHL	NBHL-BC
Glass	69.231	63.077	**75.385**	66.700	69.231
Iris	93.333	93.333	93.333	**97.400**	95.556
Pima	74.892	69.264	**77.056**	73.400	**77.056**
Teaching	52.174	52.174	60.870	55.300	**65.217**
Wine	96.296	92.593	**98.148**	80.000	**98.148**
Yeast	59.641	48.430	**61.883**	36.700	54.036
Zoo	96.774	96.774	96.774	**100.00**	**100.00**

Table 5: Classification accuracy results of the NBHL-BC compared to Multi Layer Perceptron (MLP), Decision Tree C4.5 (DT), Random Forest (RF), and Network Base High Level Classification (NBHL) using the testing dataset.

Dataset	k	e	b	α	accuracy
Wine	8	0.5	5	0.0	95.492
Wine	8	0.5	5	0.1	96.619
Wine	8	0.5	5	0.2	96.619
Wine	8	0.5	5	0.3	96.619
Wine	8	0.5	5	0.4	**97.175**
Wine	8	0.5	5	0.5	96.619
Wine	8	0.5	5	0.6	96.063
Wine	8	0.5	5	0.7	96.048
Wine	8	0.5	5	0.8	95.508
Wine	8	0.5	5	0.9	96.619
Wine	8	0.5	5	1.0	96.048

Table 6: Results of 10-folds cross validation in UCI Wine dataset with the training dataset.

5. Conclusions

In this paper, we describe a new technique for high-level classification using $betweenness\ centrality$ property. We propose that nodes with similar $betweenness\ centrality$ could determinate the new untagged instance belongs to. This measure provides the importance of each node in the sub-graph communication. We exploit this property to classify a new node into a sub-graph that presents a similar communication structure. We test this algorithm in 4 toy datasets and 7 real datasets and the results are promising.

As further works, we think that it is needed some procedures to reduce the noisy instances, and attributes that could produce disconnected sub graphs. Also, it is needed a way to detect the best parameters for $k, p, e,$ and α perhaps following an optimization approach like particle swarm.

References

Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Carneiro, M. and Zhao, L. (2018). Organizational data classification based on the importance concept of complex networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29:3361–3373.

Christiano Silva, T. and Zhao, L. (2016). *Machine Learning in Complex Networks*. Springer International Publishing.

Colliri, T., Ji, D., Pan, H., and Zhao, L. (2018). A network-based high level data classification technique. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Fadaee, S. A. and Haeri, M. A. (2019). Classification using link prediction. *Neurocomputing*, 359:395 – 407.

Freeman, L. C. (1977). A set of measures of centrality based upon betweenness. *Sociometry*, 40:35–41.

Géron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1st edition.

Murphy, K. P. (2013). *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.].

Needham, M. and Hodler, A. (2019). *Graph Algorithms: Practical Examples in Apache Spark and Neo4j*. O'Reilly Media, Incorporated.

Newman, M. E. J. (2003). Mixing patterns in networks. *Phys. Rev. E*, 67(2):026126.

Patel, A. A. (2019). Hands-on unsupervised learning using python.

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

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591 vol.1.

Shafer, J., Agrawal, R., and Mehta, M. (2000). Sprint: A scalable parallel classifier for data mining. *VLDB*.

Silva, T. C. and Zhao, L. (2012). Network-based high level data classification. *IEEE Transactions on Neural Networks and Learning Systems*, 23(6):954–970.

Silva, T. C. and Zhao, L. (2015). High-level pattern-based classification via tourist walks in networks. *Information Sciences*, 294:109 – 126. Innovative Applications of Artificial Neural Networks in Engineering.