Neurocomputing 359 (2019) 395–407

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

# Classification using link prediction

Seyed Amin Fadaee, Maryam Amir Haeri*

*Department of Computer Science and Information Technology, Amirkabir University of Technology, Iran*

ARTICLE INFO

*Article history:*
Received 21 February 2019
Revised 24 May 2019
Accepted 5 June 2019
Available online 13 June 2019

Communicated by Prof. H. Zhang

*Keywords:*
Classification
Link prediction
Graph representation
Local similarity measure
Similarity-based techniques

ABSTRACT

Link prediction in a graph is the problem of detecting the missing links or the ones that would be formed in the near future. Using a graph representation of the data, we can convert the problem of classification to the problem of link prediction which aims at finding the missing links between the unlabeled data (unlabeled nodes) and their classes. To our knowledge, despite the fact that numerous algorithms use the graph representation of the data for classification, none are using link prediction as the heart of their classifying procedure. In this work, we propose a novel algorithm called CULP (*C*lassification *U*sing *L*ink *P*rediction) which uses a new structure namely Label Embedded Graph or LEG and a link predictor to find the class of the unlabeled data. Different link predictors along with Compatibility Score - a new link predictor we proposed that is designed specifically for our settings - has been used and showed promising results for classifying different datasets. This paper further improved CULP by designing an extension called CULM which uses a majority vote (hence the M in the acronym) procedure with weights proportional to the predictions' confidences to use the predictive power of multiple link predictors and also exploits the low level features of the data. Extensive experimental evaluations shows that both CULP and CULM are highly accurate and competitive with the cutting edge graph classifiers and general classifiers.



## 1. Introduction

Classification is an old problem in machine learning and pattern recognition that aims at finding a correct mapping between data and their corresponding labels. This mapping would then be used to derive the class of the unlabeled data [1].

This field is still highly active in the literature and a lot of algorithms have been proposed to correctly classify the data. Most of the classification algorithms aim at finding a decision boundary in the feature space for distinguishing the data belonging to different classes; however, as more complex data require more complex algorithms, these approaches could fail or not capture the true relations in the data.

One of the new approaches that has recently gained popularity in the literature is classification of the unlabeled instances using the graph representation of the data. Data can be represented in different forms one of which is a graph. In this setting, the data is first converted to a graph via a similarity function in the feature space, then unlabeled data is classified by incorporating a graph property. These graph properties are called *high* level feature which give more insight to the data compared to the *low* level features.

Classification using graph representation is studied extensively in numerous works [2–9]. These works use graph properties such as clustering coefficient, modularity, importance, PageRank and others to classify the unlabeled data and they tend to achieve more accurate results compared to the classifiers that classify based on the low level features of data. This approach has been used in text classification [10], hyperspectral image classification [11,12], image classification [2,8], handwritten digits recognition [3] and other areas.

Link prediction is the problem of predicting the missing link in a graph or the ones that would be formed in the near future [13]. Using the graph representation of the data we can treat the classification as a link prediction problem in an intuitive way where we try to find the link between the unlabeled node with it corresponding class. To our knowledge, there are not any work in the literature that uses link prediction to solve the problem of classification, however, the use of classification to solve link prediction is studied extensively [13].

In this work, we proposed an algorithm called *CULP* (acronym for Classification Using Link Prediction) that takes a different look at the classification problem through a link prediction approach. As we will elaborate in the paper, CULP uses a graph called LEG that models the data in an intuitive and suitable way for link prediction.

Any link predictors can be used to derive the class of the unlabeled node in CULP and we proposed a new local measure called

* Corresponding author.
*E-mail addresses:* aminfadaee@aut.ac.ir (S.A. Fadaee), haeri@aut.ac.ir (M. Amir Haeri).

https://doi.org/10.1016/j.neucom.2019.06.026
0925-2312/

*Compatibility Score* that is designed to improve the accuracy of link prediction and consequently classification.

As much insight as high level features have for capturing the patterns present in the data, exploiting the low level feature alongside them would further improve the predictive power of a graph classifiers and different researchers incorporate this idea in their work [2,4]. This is why we further improved CULP and proposed the *CULM* extension - a majority vote system (hence the *M* in the acronym) with weights proportional to the probabilities of the predictions, this extension uses multiple link predictors along with a low level classifier. As we will see both CULP and CULM algorithms derive highly accurate results which are competitive with low level classifiers and other graph based classification methods.

The rest of the paper is organized as follows; in the next section a review of the general domains used in this paper is presented which is a preliminary section elaborating the problem of link prediction, similarity measures in vector space, method of converting graph to data and the problem of classification. After that a section of related works is given which is a summary of recent works using graph representation of the data for classification. Next, the CULP algorithm is presented with full details which elaborates on the *LEG* (Label Embedded Graph) structure, the classification procedure which uses link prediction, our novel link predictor - Compatibility Score, the time complexity and a toy example to demonstrate CULP. Finally, the CULM extension is presented which is followed by our extensive experimental results to put our proposed algorithms into perspective. At the end, the conclusion to the paper and the aim for future works are presented.

## 2. Preliminaries

To fully understand CULP, a grounding for the details comprising this algorithm should be set. In this section, a general review to graph theory concepts and notations along with the definition of the link prediction problem in complex networks is given. After that, an overview of some of the most important similarity measures is presented, following this the different ways of converting data to graph is discussed. Finally at the end of this section the problem of classification is defined.

### 2.1. Link prediction

Given a set of vertices $V$ and a set of edges $E$ containing $(i, j)$ where $i, j \in V$ the data structure $G(V, E)$ can be defined as a graph. If the elements in $E$ are ordered pairs, $G$ is considered to be a *directed graph*. In an *undirected graph* if $(i, j) \in E$ it is implied that $(j, i) \in E$. Regardless of the directionality of the graph, node $j$ is a *neighbor node* to node $i$ if $(i, j) \in E$. For a node $i$, $\Gamma_i$ is the set of the neighbor nodes of $i$.

For the graph $G$, adjacency matrix $A_G$ or simply $A$ is defined as an $N \times N$ matrix with zero-one elements and $N = |V|$. For any entry in $A$, $A_{i,j} = 1$ if and only if $(i, j) \in E$. In an undirected graph by definition $A = A^T$. As our focus in this paper is toward undirected graph, for the sake of simplicity we use *graph* to state an undirected graph.

The degree of a node $i$ in a graph can be derived using $|\Gamma_i|$. For any graph, the cardinality or $|E|$ can be obtained by summing over the degree of all nodes using Eq. (1) where $N = |V|$.

$$|E| = \frac{1}{2}\sum_{i=1}^{N} |\Gamma_i| \tag{1}$$

The problem of link prediction in a graph arises when the goal is to predict for the currently absent links (0 entries in $A$) the probability of link formation in the future. There are many functions to predict the link prediction scores. These functions usually compute the local similarity between the nodes to derive the scores. One of the simplest techniques is known as *common neighbors* (CN) [14]. Using this approach the prediction scores can be derived using the following:

$$\lambda_{i,j} = |\Gamma_i \cap \Gamma_j| \tag{2}$$

Eq. (2) simply counts the number of common neighbors of nodes $i$ and $j$ to derive a score for their link formation.

Another approach to find the link formation score is introduced by Adam and Adar [15] which uses degrees of common neighbors as features for prediction and it can be written as

$$\lambda_{i,j} = \sum_{\gamma \in \Gamma_i \cap \Gamma_j} \frac{1}{log|\Gamma_\gamma|} \tag{3}$$

Eq. (3) is known as the *Adamic-Adar score* (AA). This score penalizes the features by their logarithm and uses these features for deriving the prediction scores. Another famous approach for tackling the problem of link prediction is the *Resource Allocation Index* (RA) [16] that simulates the transition of resources between nodes $i$ and $j$. This index is defined as Eq. (4).

$$\lambda_{i,j} = \sum_{\gamma \in \Gamma_i \cap \Gamma_j} \frac{1}{|\Gamma_\gamma|} \tag{4}$$

This index is quite similar to AA, however it does not use the logarithm function which reduces the effect of nodes with high degree. This has the benefit of penalizing high degree common nodes. In a lot of networks, these nodes provide little insight for link prediction as they are connected to a lot of other nodes in the graph.

In this work, we are proposing a new similarity function used for the purpose of link prediction. called *Compatibility Score* which is discussed further in the paper.

### 2.2. Similarity measures

Any data point $x$ with numeric features $x_f$ where $1 \le f \le d$ can be regarded as a vector in an $d$-dimensional space. This view would enable the measurement of the similarities between data points using conventional similarity measures. As we are going to utilize a similarity measure in converting our data to graph(discussed in the next segment), we are going to provide overview of some of these measures.

Having our data matrix $X$, with $n$ rows and $d$ columns with each row being a data vector, the *Cosine* similarity can be defined as the following:

$$s_{i,j} = \frac{X_i . X_j}{\|X_i\|_2 \left\|X_j\right\|_2} \tag{5}$$

where $\|x\|_2$ denotes the Euclidean norm of the vector x which is derived by the following:

$$\|x\|_2 = \sqrt{\sum_{f=1}^{d} x_f^2}$$

Following the above equation, the Euclidean distance between any two $d$ dimensional vectors can be written as:

$$\phi_{i,j} = \sqrt{\sum_{f=1}^{d} (X_{i,f} - X_{j,f})^2} \tag{6}$$

Utilizing the Euclidean distance, another similarity measure - namely *Inverse Euclidean* can be defined using:

$$s_{i,j} = \frac{1}{\phi_{i,j} + \epsilon} \tag{7}$$

In Eq. (7) the $\epsilon$ term is a small number used to avoid division by zero in case of identical vectors. Another prominent distance in linear algebra is what is known as the absolute or Manhattan distance (Eq. (8)) and by substituting Eq. (8) in Eq. (7), the *Inverse Manhattan* similarity function is defined.

$$\phi_{i,j} = \sum_{f=1}^{d} |X_{i,f} - X_{j,f}| \tag{8}$$

### 2.3. Converting data to graph

Any vector based data can be represented as a graph. Doing this would result in changing the structure of the data which enables us to compute high level features.

Two of the most used procedures for converting data to graph are $r$-Radius and $k$NN methods [17].

Using a similarity measure (e.g. cosine similarity discussed in the previous segment) $s$ and matrix data $X$ we can use either of these two algorithms to convert the data into a graph. In $r$-Radius, an edge is created between every pair of data points that have a similarity higher than a predefined threshold $r$. Another approach is using $k$-nearest neighbors to form up the graph. If (based on a $s$) $X_i$ is in the $k$-nearest neighbors of $X_j$ the edge $(i, j)$ is created.

Due to the fact that $k$NN relation is not symmetric this approach would generally results in a directed graph. However the same principle can be used to create an undirected graph as in Algorithm 1. Using this approach, if $X$ has $N$ instances, the number of undirected edges $|E|$ in the created graph is bounded by $\frac{Nk}{2} \leq |E| \leq Nk$. CULP uses an undirected $k$NN modeling of the data for the task of classification.

**Algorithm 1** Undirected $k$NN conversion function for the data matrix $X$ and similarity measure $s$.

```
function kNN-Convert(X, s, k)
    E = {}
    for i, j ∈ X do
        if i ∈ kNN(s, j) or j ∈ kNN(s, i) then
            E ← E ∪ (i, j)
        end if
    end for
    return E
end function
```

### 2.4. Classification

Suppose there are two sets of data, $X$ with $n$ instances and $d$ features for each instance which is the set of our labeled data. The labels of $X$ is denoted by $y$ where $y_i \in 1, 2, \ldots, C$ with $C$ being the number of classes. Each pair $(X_i, y_i)$ makes up our training data. The other set of data is $X^{(u)}$ with $m$ instances and again $d$ features for each instance which are the unlabeled or the test data.

The classification problem aims at finding a mapping $X_i^{(u)} \rightarrow \hat{y}_i$ for every $i \in 1, \ldots, m$. In other words, we are trying to find a proper label for each of the unlabeled instance in $X^{(u)}$. If $C = 2$, this is called *binary classification* and if $C > 2$, the problem is called *multi-class classification* [1].

Classifiers like $k$NN or *Decision Tree* can naturally handle multi-class classification problems, however some classifiers like *SVM* are inherently designed for the binary classification task and upgrading them to handle multi-class classification requires using *One vs. All* or *One vs. One* approaches [1].

In one vs. all, $C$ classifiers are trained and each classifier has the task of deciding whether an instance belongs to a particular class or not. The one vs. one approach is done by training $C(C-1)/2$ classifiers to classify an instance into either of two classes among all of the $C$ classes.

## 3. Related works

Using graph classification has recently gained popularity and numerous works [2–8] focus on using this approach instead of the classical methods of classification. These method can capture complex patterns in the data and they can generate high level features to guide the classification procedure, furthermore they can usually be modified to utilize the low level features of the data as well.

In [2] a random walker is used to classify unlabeled instances on the graph embedding of the data. This graph is represented by a weight matrix of similarities. The random walk process is continued until convergence and the new data receives the label through a weighted majority vote between the labels of the top $\eta$ nodes with highest probabilities. This method takes the similarity among the data points into account with a single network for the dataset along with structural changes of an unlabeled instance on the networks created for each class. The complexity of the method is of $O(n^2)$, however, as the authors claimed, using sparse representations such as $k$NN network, and graph construction method based on Lanczos bisection [18], this complexity can be reduced to a complexity between $O(n^{1.06})$ and $O(n^{1.33})$.

Another system is proposed in [9] in which a graph is created for the training instances of each class, then using the proposed *spatio-structural differential efficiency* measure in the paper, a test instance is connected to some of the nodes in each graph. The label of the data would be the class of the graph that the test data has the highest importance in. The importance is characterized by Google's PageRank measure of the network. The spatio-structural differential efficiency measure in [9] takes considers both physical and topological properties of the data and the complexity of the proposed method is again of $O(n^2)$ which is once more reduced to a complexity between $O(n^{1.06})$ and $O(n^{1.33})$ by using graph construction method based on Lanczos bisection.

A hybrid method is proposed in [3] that aids a typical classifier (such as *kNN, SVM* or *Naive Bayes*) by using high level features. These high level features are the difference of some graph properties before and after inserting a new instance into the graph representation of the data of each class. The graph of each class is constructed using combination of $r$-radius and $k$NN graph conversion methods. The graph properties used in their work are *assortativity, network clustering coefficient* and *average degree*. The label for the test instance is generated by a weighted combination of low level and high level features. The authors extended their work in [4] by using two more high level features namely *Normalized Average Distance among vertices* and *coreness variability* and using a stacking procedure to learn the weight for each feature. Also [5] extends the same work by discarding the use of any classical classifier and using a scheme that takes low level features techniques into account to filter irrelevant graphs of some of the classes.

Authors of [6] proposed a framework for classification using *k-Associated Optimal Graph* for modeling the data and *Bayes theorem* and computing a posterior probability for each class to classify new instances. Similar to $k$NN graph conversion method, k-Associated Optimal Graph computes the similarity of a data point with all of the training data, however, it would form an edge only if the points belong to the same class. This would result in having multiple component (and possibly more than one component for a class). The method furthermore tries to find a local $k$ for each class so that the resulting components get the maximal *Purity* (a measure based on average degree of a component). This way the process of finding the parameter $k$ is conducted automatically which also make the complexity of the framework of $O(n^2)$. Another paper

[3] also uses the k-Associated graph in this paper along with the high level classification method of [3] to classify new instances.

Other methods using different graph measures have been produced as well. Neto and Zhao [7] uses *dynamic entropy* for each weighted graph produced by *r*-radius where the weights denote the distance between data points. Cupertino et al. [8] utilizes the *modularity* measure for classifying new instance that belongs to a pattern set of the same object in the training data. The label is derived by creating a *k*NN graph for each pattern set and choosing the label of the graph with lowest modularity change after insertion of the new data. Both of the methods in [7,8] have the complexity of $O(n^2)$.

The graph based classification methods in the literature mostly have three characteristics in common. Firstly they create a different graph for each classes of the data; this approach avoids finding meaningful pattern that may form by the similarities between points in different classes.

The second aspect these algorithms have in common is that they treat test instances individually and add them to the graph of each class and measure a graph property before and after the insertion. This makes the prediction of a new instance inefficient in presence of large amount of test data.

Lastly, the properties that these algorithms use for finding the differences before and after the insertion of the unlabeled data (e.g. clustering coefficient, average path etc.) are time consuming and their computation times are usually dependent on the graph size which can make them infeasible for large datasets.

Our proposed algorithm CULP and it's extension CULM solves the first and second issue by employing a novel graph representation called *LEG* which treats classes as nodes along with training and test instances as a unified object and is discussed further in the paper. As for the third problem, since the label of a test instance is derived using link prediction measures (as discussed in the previous section), the classification of the unlabeled data is faster than the similar methods.

## 4. CULP algorithm

*CULP* (**C**lassification **U**sing **L**ink **P**rediction) is a classification method aimed to gain a higher accuracy in mulit-class classification task by exploiting the similarity among the data points. This algorithm employs the powers of graph representation and link prediction methods in complex networks to deal with this problem.[1] The overall structure of CULP is consisted of 2 stages:

1. Creating the LEG structure $G$ from the data
2. Classifying the test data using $G$

In the first step we model our data into an augmented graph data structure called *LEG* (**L**abel **E**mbedded **G**raph) which we call *G*. $G$ is a heterogeneous graph which incorporates the data, the classes and the similarity between them as a unified object.

LEG essentially contains 3 sets of nodes and 2 sets of links. The different type of nodes in $G$ are training nodes, testing nodes and class nodes, also a link between two data nodes denotes similarity between them and a link between a training node and a class node denotes the class membership of that node.

After creating $G$, we can convert the classification problem to the problem of predicting the class membership link of a testing node. By utilizing a link prediction algorithm in the next step, membership score for every testing-class pair of nodes is computed.

Each of the membership scores acts as a posterior probability. A label is chosen for a testing node based on these scores.

[1] The complete code of CULP in python can be found in github.com/aminfadaee/culp.

CULP procedure is depicted in Algorithm 3. In the next segments each of the steps of the proposed algorithm is covered in more detail.

### 4.1. LEG representation

The first step toward classification using CULP is creating the LEG representation. LEG is a heterogeneous graph with three sets of nodes:

- Training nodes ($V_l$)
- Testing nodes ($V_u$)
- Class nodes ($V_c$)

and two sets of edges:

- Similarity edges ($E_s$)
- Class membership edges ($E_c$)

Each set of nodes correspond to their analogous set of data i.e. $V_l$ contains $n$ nodes, $V_u$ contains $m$ nodes and $V_c$ contains $C$ nodes.

The class membership edges are created based on the labeled data. $E_c$ contain edges $(i, j)$ where $i \in V_l$ and $j$ is the node representation of $y_i$, meaning that each training node is connected (without direction) to its corresponding class node. It should be noted that since the labels for the test data is not available, $E_c$ contains only pair of nodes from $V_l$ and $V_c$.

Unlike $E_c$, the members of $E_s$ are not obtained so trivially. $E_s$ is responsible for incorporating the similarities between instances of our data and the edges in this set are obtained by using a graph conversion algorithm. In this work the undirected version of *k*NN graph conversion (Algorithm 1) is used.

Edges in $E_s$ primarily connect two nodes in $V_l$ or a node from $V_u$ to one in $V_l$. However, there is no constraint on having an edge between two nodes in $V_u$, meaning that we can find the similarity between unlabeled data and connect them as well (as we have done in this work).

If the unlabeled data is not available at first or in case of a new unlabeled node $x^{(u)}$ this node is first added to the set $V_u$, after that the similarity edges between this node and other nodes of the graph is created through a linear similarity computation.

After creating all of the sets of nodes and edges, we can define the LEG $G(V, E)$ where $V = V_l \cup V_u \cup V_c$ and $E = E_s \cup E_c$. Although $G$ is inherently heterogeneous, we can treat it as a simple undirected graph. The procedure for creating $G$ is summarized in Algorithm 2.

**Algorithm 2** LEG construction function for the data $X^{(l)}$, the labels $y$ and the unlabeled data $X^{(u)}$ with parameter $k$ and the similarity function $s$.

**function** LEG($X^{(l)}$, $X^{(u)}$, $y$, $s$, $k$)
  $X = X^{(l)} \cup X^{(u)}$
  $V_l \leftarrow \{1, 2, \ldots, n\}$//Nodes are represented by numbers
  $V_u \leftarrow \{n+1, n+2, \ldots, n+m\}$
  $V_c \leftarrow \{n+m+1, n+m+2, \ldots, n+m+C\}$
  $E_c \leftarrow \{\}$
  **for** $i \in \{1, 2, \ldots, n\}$ **do**
    $E_c \leftarrow E_c \cup (i, n+m+y_i)$
  **end for**
  $E_s \leftarrow$kNN-CONVERT$(X, s, k)$
  $V \leftarrow V_l \cup V_u \cup V_c$
  $E \leftarrow E_s \cup E_c$
  **return** $G(V, E)$
**end function**

This algorithm takes the labeled and unlabeled data along with the parameter $k$ and the similarity measure $s$ and produces $G$ as the output.

There are always $n$ edges belonging to $E_c$. The number of edges in $E_s$ however, has an upper and lower bound. The minimum number of possible edges in $E_s$ is obtained when the $k$NN procedure of each pair of points in $X$ ($X^{(u)} \cup X^{(l)}$) is symmetric - meaning that $\forall i \forall j,\ i \in kNN(j) \leftrightarrow j \in kNN(i)$. The maximum number of edges in $E_s$ on the other hand is obtained when the $k$NN procedure is *not* symmetric for any pair of nodes in $X$. Using these, the bounds on the number of edges in a LEG can be derived as Eq. (9).

$$n + \frac{k}{2}(n+m) \leq |E| \leq n + k(n+m) \tag{9}$$

By the bounds in Eq. (9), it can be stated that $G$ gives us a new low memory cost representation of the data. The memory for the original data is of $O(n \times d + m \times d + n)$ for $X^{(l)}$, $X^{(u)}$ and $y$, but since it is usually the case that $k << d$ for high dimensional data, LEG saves a lot of memory compared to using the original data for the task of classification.

Another aspect of LEG is the fact that we are incorporating all of our labeled and unlabeled data and class labels in a unified structure that enables us to find the labels of the test data via simple and efficient graph properties, specifically link prediction methods which is covered in the next segment.

### 4.2. Classification

As stated before, in classification, the goal is to find a mapping $X_i^{(u)} \rightarrow \hat{y}_i$ for every $i \in 1, \ldots, m$. Using the LEG representation, this problem can be reformatted as finding $j^*$ for $\forall i \in V_U$ so that the probability of $(i, j^*) \in E_c$ is maximized.

The new formulation means that edges will be added to the set $E_c$ by predicting the most probable membership link for every test node. This can be easily done via link prediction methods discussed before.

Using a local similarity measure $\lambda$ for link prediction (e.g. Adamic-Adar index), this problem can be solved using the following:

$$\begin{cases} \forall i \in V_u, \quad E_c \leftarrow E_c \cup (i, j^*) \\ j^* = \underset{j \in V_c}{argmax}(\lambda_{i,j}) \end{cases} \tag{10}$$

Although more complex link prediction methods (random walk, average path length etc.) can be used to solve the problem, the local similarity measures are not only extremely fast and efficient to compute but they also derive competitively accurate results as it will be discuss in the experiments. The pseudocode of CULP is depicted in Algorithm 3.

**Algorithm 3** CULP Algorithm.

```
function CULP(X, X^(u), y, s, k, λ)
    G ← LEG(X, X^(u), y, s, k)
    ŷ ← {}
    for i ∈ V_u do
        j* ← argmax_{j∈V_c}(λ_{i,j})
        ŷ_i ← j* − (n + m)
    end for
    return ŷ
end function
```

### 4.3. Compatibility score

In this work a novel local score for link prediction is formed which is designed specifically for the task of classification. This new similarity function is called *Compatibility Score* and like Adamic-Adar and Resource Allocation scores penalizes the common neighbors, however, this penalization is done differently.

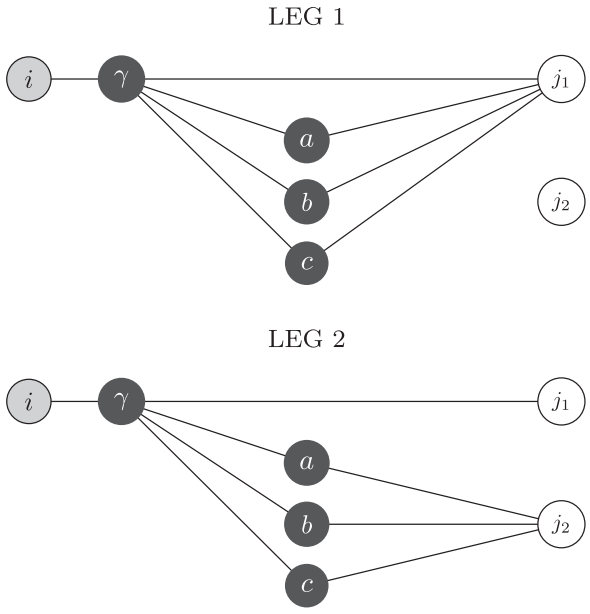


**Fig. 1.** Using AA or RA for predicting the formation of $(i, j_1)$ in both LEG's would result in the same score, however node $\gamma$ in the first case is more valuable for the prediction.

Both AA and RA scores can be unfair in some instances, meaning that they can over-penalize a valuable common neighbor or give the same score to two inherently different nodes. Take the two LEGs in Fig. 1 for example ($i \in V_u$, $\gamma$, $a$, $b$, $c \in V_l$ and $j_1, j_2 \in V_c$). In both cases the goal is to find the score for the $(i, j_1)$ link. AA and RA would both penalize node $\gamma$ in the same way (penalty of 5 for RA and $log(5)$ for AA); however, in the first LEG the node $\gamma$ is more valuable than that of the second LEG and this is due to the fact that three neighbors of this node ($a$, $b$, $c$) are also connected to node $j_1$.

When trying to predict the score for the formation of link between nodes $i$ and $j$ with a common neighbors between them namely $\gamma$, two sets of edges can be defined starting from $\gamma$: *compatible edges* and *incompatible edges*.

Compatible edges for node $\gamma$ are the ones connecting $\gamma$ to nodes which are by themselves connected to the destination of the candidate link ($j$ in this case). We can define incompatible edges as all the other edges which are not compatible.

Now the cardinality of incompatible edges or the *incompatibility penalty* for node $\gamma$ which is a common neighbor of nodes $i$ and $j$ can be defined as the following:

$$\delta(i, j, \gamma) = |\Gamma_\gamma| - |\Gamma_\gamma \cap \Gamma_j| \tag{11}$$

Using Eq. (11) the Compatibility Score (*CS* for short) is formally defined as Eq. (12). In this equation both $\delta(i, j, \gamma)$ and $\delta(j, i, \gamma)$ are used for the prediction of $(i, j)$ to make the score symmetric so that $\lambda_{i,j} = \lambda_{j,i}$.

$$\lambda_{i,j} = \sum_{\gamma \in \Gamma_i \cap \Gamma_j} \frac{1}{\delta(i, j, \gamma)} + \frac{1}{\delta(j, i, \gamma)} \tag{12}$$

Using the Compatibility Score for the cases of Fig. 1 the score for link $(i, j_1)$ in LEG 1 can be computed as 0.7 and in LEG 2 as 0.4. This is the desired outcome as the score in LEG 1 is now higher. In the experiments, a more detailed comparison of CS with other link prediction methods is done.

### 4.4. Time complexity analysis

In this subsection, the time complexity of finding the class membership edge of a test node will be analyzed. The main component in finding the correct link is the local similarity measure $\lambda$ which is used for link prediction. These local measures find the score in time proportional to the degree of their source and destination nodes. In CULP, the source node $i$ belongs to $V_u$ and the destination node $j$ belongs to $V_c$. So the first step in analyzing the time of finding a class membership edge is finding the average degree of nodes in $V_u$ and $V_c$.

The degree of node $j$ is the number of labeled nodes connected to it or more specifically $n_j$ which is the number of data points with class of node $j$; however, for the degree of $i$ a more detailed analysis is needed. As stated before, in any undirected graph Eq. (1) holds. Eq. (1) can be rewritten as the following:

$$|E| = \frac{1}{2}\left(\sum_{i\in V_c}|\Gamma_i| + \sum_{i\in V_l}|\Gamma_i| + \sum_{i\in V_u}|\Gamma_i|\right) \tag{13}$$

Since the degree of the class nodes sums up to the number of labeled data $n$, it can be substituted in the above equation; on the other hand, if we treat each node in $V_u$ to have average degree $D$, we can state that nodes in $V_l$ would have average degree of $D+1$ (since each of them has also a membership edge). Using all these, the above formula can be rewritten in the following manner:

$$|E| = \frac{1}{2}(n + n(D+1) + mD)$$

$$|E| = n + \frac{nD}{2} + \frac{mD}{2} \tag{14}$$

As stated before the number of edges in a LEG is bounded by an upper and lower bound which is derived in Eq. (9). Now using Eqs. (14) and(9) the upper bound of $D$ can be defined as:

$$n + \frac{nD}{2} + \frac{mD}{2} = k(n+m) + n$$

$$D = 2k \tag{15}$$

and its lower bound as:

$$n + \frac{nD}{2} + \frac{mD}{2} = \frac{k}{2}(n+m) + n$$

$$D = k \tag{16}$$

Consequently, the average degree for labeled and unlabeled nodes is of $O(k)$ and for class nodes is of $O(n)$. The *Common Neighbor, Adamic-Adar* and *Resource Allocation* all have the complexity of finding the common neighbors between source and destination which is the intersection of the neighborhoods of the two nodes. The *Compatibility Score* however, first finds the common neighbors and does two intersection for each of the nodes in the common neighbor set.

If done efficiently, the intersection of two sets with sizes $a$ and $b$ can be obtained in order of $O(min(a, b))$ in average. Using this, the complexity of finding the score in LEG for the formation of links between $i$ and $j$ is of $O(k)$ when *Common Neighbor, Adamic-Adar* or *Resource Allocation* is used and is $O(k^2)$ when *Compatibility Score* is used. Since $k$ is usually small (in our experiments $1 \le k \le 35$), it is safe to state that the link prediction is done in constant time; also as there are $C$ nodes in $V_c$, predicting the label of $m$ instances would take time of $O(mC)$ after creating the LEG.

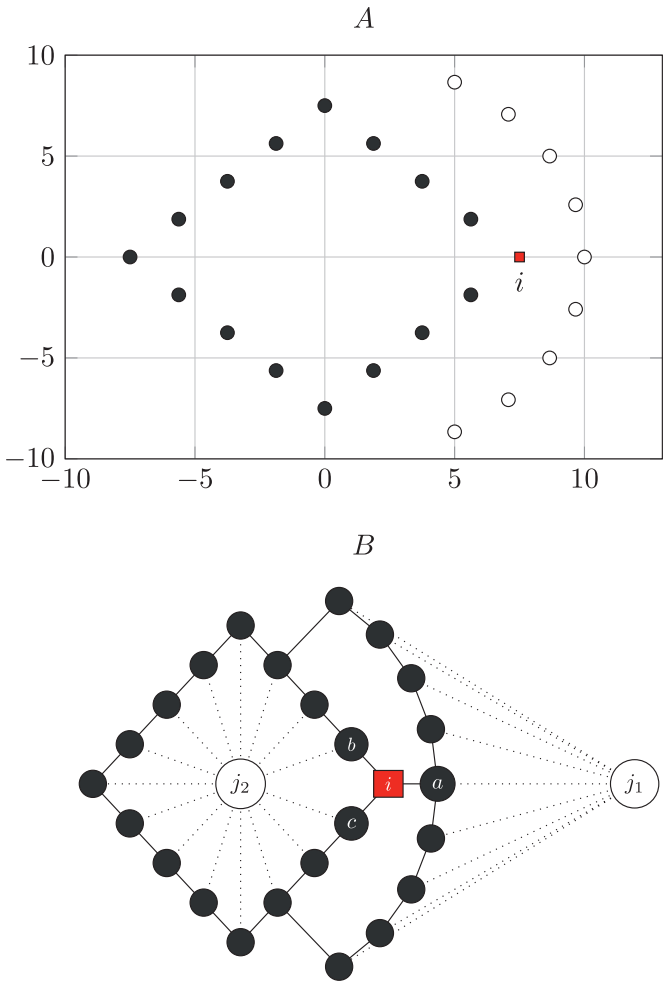


**Fig. 2.** Toy example demonstrating CULP. A- The set of data belonging to 2 classes and a test point in red B- LEG graph of the data.

### 4.5. Toy example

In this subsection a simple classification problem is solved using CULP to demonstrate the steps involving in this algorithm. The data is presented in Fig. 2-A as two classes. The white points represent the data of class 1 and the dark points belong to class 2. The problem is finding the correct label of the red point (point $i$).

The first step is choosing a similarity function $s$ and a value for the parameter $k$ for forming the graph. Here we chose $k = 2$ and the Euclidean similarity (discussed in the preliminaries section).

Now the node sets can be defined as $V_c = j_1, j_2$. $V_u = i$ and all the other points as the set $V_l$. By creating the edges in $E_c$ and $E_s$ as shown in Algorithm 2 the LEG in Fig. 2-B can be derived. As can be seen, in this graph every node except for $i$ is connected to one of the class nodes $j_1$ and $j_2$ (white nodes) by dotted links and the black links represents the edges of $E_s$.

Looking at the graph, it can be seen that the node $i$ is connected to nodes $a$, $b$ and $c$. This means these nodes would assist in finding the label for node $i$. Using these nodes, the scores for edges $(i, j_1)$ and $(i, j_2)$ can be obtained with each of the scores discussed before as $\lambda$. The results of computing these scores are depicted in Table 1.

The results of all the link predictors in Table 1 show that the score for the link $(i, j_2)$ is higher. This prediction matches the pattern perceived by looking at the data in Fig. 2-A and is the correct prediction.

**Table 1**
Scores computed by 4 different link predictor for the toy example of Fig. 2.

| $\lambda$ | $(i, j_1)$ | $(i, j_2)$ | Prediction |
|---|---|---|---|
| CN | 1 | 2 | 2 |
| AA | $1/log(4)$ | $2/log(3)$ | 2 |
| RA | 1/4 | 2/3 | 2 |
| CS | $1/2+1/4$ | $2(1/2+1/3)$ | 2 |

## 5. CULM extension

As we stated in the time complexity analysis subsection and demonstrated in the toy example of the previous section, once the LEG structure is formed, the prediction of links can be done instantly; knowing this and the fact that there are different options in choosing the link predictor $\lambda$, the question arises as to *why not use all of our predictors and somehow combine their predictive capabilities to assist us in finding the best membership link for a test node?*

The next question arises after we analyze the related works done in the field of classification using complex network representations. A good portion of these methods are capable of incorporating or exploiting the low level features of the data to enhance the classification performance. *How can we modify our framework CULP to exploit the low level features of the data as well as the high level features?*

The answer to both of these questions lies in our extension to CULP algorithm which we call the *CULM* extension. CULM increases the predictive capabilities of CULP by using a *weighted majority vote* procedure (hence the **M** as in **M**ajority in the end instead of **P**).

Instead of using only one link predictor $\lambda$, we will use an array of link predictors $\Lambda$. Each link predictor $\lambda$ when used, gives a score to the links $(i, j)$ for all $j \in V_c$. We can use all of these scores to estimate the probability $p$ of our prediction correctness as Eq. (17).

$$p_{\hat{y}} = \frac{\lambda_{i,j^*}}{\sum_{j \in V_c} \lambda_{i,j}} \tag{17}$$

In this equation $\hat{y}$ is the label corresponding to $j^*$ and $j^*$ is computed using Eq. (10) of the previous section. Using Eq. (17) we can assign confidence to the prediction of $\lambda$. When using multiple predictors, it is obvious that a $\lambda$ with higher confidence is more reliable. We are going to use these probabilities to assign weights to each of the $\lambda$s in $\Lambda$. This way instead of using a simple majority vote, a weighted voting procedure can be used. In a weighted majority vote procedure, few predictions are aggregated. Each of these prediction has an individual weight which states the value of their vote; finally the voting in this setting would be done as Algorithm 4.

**Algorithm 4** Weighted Majority Voting Algorithm.

```
function VOTE(Y, W)
    L ← {0}_C
    for y ∈ Y and w ∈ W do
        L_y ← L_y + w
    end for
    ŷ ← argmax(L)
    return ŷ
end function
```

In Algorithm 4, $Y$ is the set containing the predicted labels of each of the predictors, $W$ is the respective weights of the labels and $L$ is a set with $C$ elements which keeps track of the weight for each of the classes. Using this algorithm enables us to not only use multiple link predictors' predicted labels, but also incorporate arbitrary any classical classifier $\psi$ with suitable weights. This way the low level features of the data is exploited as well.

The next step is to define the weights for each of our predictors and $\psi$. If $\hat{y}_\lambda$ is the predicted label of the predictor $\lambda$ for the unlabeled data $x^{(u)}$ and $p^\lambda_{\hat{y}}$ is the probability of this prediction, the weight of predictor $\lambda$ for $x^{(u)}$ can be defined as Eq. (18). Also for the prediction of $\psi$ on $x^{(u)}$ which can be denoted as $\hat{y}_\psi$, we can define the weight as Eq. (19).

$$w^\lambda_{\hat{y}} = \frac{\alpha p^\lambda_{\hat{y}}}{\sum_{\lambda' \in \Lambda} p^{\lambda'}_{\hat{y}}} \tag{18}$$

$$w^\psi_{\hat{y}} = 1 - \alpha \tag{19}$$

The $\alpha$ parameter which is used in both equations is provided by the user. This parameter controls the trade-off that CULM will make between the link predictors' labels and the prediction of the low level classifier.

The parameter $\alpha$ is chosen in the range 0 to 1; however any value below 0.5 would result in neutralizing the vote of CULM predictors. Also if $\alpha = 1$, the prediction is completely done by CULM predictors and the low level classifier is ignored; so in general it can be stated that $0.5 \leq \alpha \leq 1$.

Now the CULM extension can be formally defined as the procedure captured in Algorithm 5. In this algorithm, after creating the LEG, each of the predictors in $\Lambda$ produce a label and a probability. These probabilities and labels are then merged with that of the low level classifier $\psi$ to form up $Y$ and $W$ which are passed to Algorithm 4 to produce the final label for the test instance.

**Algorithm 5** CULM Algorithm.

```
function CULM(X, X^(u), y, s, k, Λ, ψ, α)
    G ← LEG(X, X^(u), y, s, k)
    ŷ ← {}
    for i in V_u do
        P ← {}
        Ŷ ← {ψ(X_i^(u))}
        W ← {1 − α}
        for λ in Λ do
            j* ← argmax_{j∈V_c}(λ_{i,j})
            P ← P ∪ λ_{i,j*} / Σ_{j∈V_c} λ_{i,j}
            Ŷ ← Ŷ ∪ j* − (n + m)
        end for
        for p ∈ P do
            W ← W ∪ α×p / Σ_{p'∈P} p'
        end for
        ŷ_i ← VOTE(Ŷ, W)
    end for
    return ŷ
end function
```

As analyzed, the time complexity of predicting the labels of $m$ instances using CULP is $O(mC)$. CULM inherently repeats the prediction $l$ times with $l$ being the number of link predictors in $\Lambda$ and then uses a majority vote. The predictions complexity is $O(lmC)$ and the voting has the complexity of $O(l)$; Therefore, we can identify CULM time complexity to be of $O(lmC + l + O(\psi))$ with $O(\psi)$ part being the complexity of the low level classifier. Clearly the general time for CULM could be majorly different upon using different classifiers.

**Table 2**
Datasets used in deriving the results for CULP and CULM.

| Dataset | Instances | Attributes | Classes |
|---|---|---|---|
| Zoo | 101 | 16 | 7 |
| Hayes | 132 | 4 | 3 |
| Iris | 150 | 4 | 3 |
| Teaching | 151 | 5 | 3 |
| Wine | 178 | 13 | 3 |
| Sonar | 208 | 60 | 2 |
| Image | 210 | 19 | 7 |
| Glass | 214 | 9 | 6 |
| Thyroid | 215 | 5 | 3 |
| Ecoli | 336 | 7 | 8 |
| Libras | 360 | 90 | 15 |
| Balance | 625 | 4 | 3 |
| Pima | 768 | 8 | 2 |
| Vehicle | 846 | 18 | 4 |
| Vowel | 990 | 10 | 11 |
| Yeast | 1,484 | 8 | 10 |
| RedWine | 1,599 | 11 | 6 |
| Segment | 2,100 | 19 | 7 |
| Optical | 5,620 | 64 | 10 |
| Poker | 25,010 | 10 | 10 |

## 6. Experimental results

In this section, we are presenting the result of our proposed algorithms CULP and CULM on 20 different real datasets and comparing it to classical classification methods as well as best classifiers of the related works in the domain of classification using complex networks.

The datasets used for our experiments are all obtained from UCI machine learning repository [19]. These datasets include *Zoo, Hayes-Roth* (Hayes), *Iris, Teaching Assistant Evaluation* (Teaching), *Wine, Sonar Mines vs. Rocks* (Sonar), *Image Segmentation* training set (Image) and testing set (Segmentation), *Glass Identification* (Glass), *Thyroid Disease* (Thyroid), *Ecoli, Libras Movement* (Libras), *Balance Scale* (Balance), *Pima Indians Diabetes* (Pima), *Statlog Vehicle Silhouettes* (Vehicle), *Vowel Recognition* (Vowel), *Yeast, Wine Quality Red* (RedWine), *Optical Recognition of Handwritten Digits* (Optical), *Poker Hand* (Poker). Each of these datasets along with the number of instances, attributes and classes is listed in Table 2.

### 6.1. CULP analysis

The reason behind choosing these datasets is the variety of both structure and domain between them. The size of these data is between 101 to 25,010 which test the practicality of our algorithms on both small and large datasets; the number of attributes vary from 4 to 90 which test the proposed algorithms against both low and high dimensional datasets and finally there is a lot of variety in the number of classes in the datasets which ranges from 2 up to 10.

This section is organized as follows: first, the experiment on CULP and different predictors as $\lambda$ is presented, after that the CULM algorithms is analyzed with 3 different low level classifier, the following subsection will discuss the effects of $\alpha$ parameter, after that a comparison of CULP and CULM with classical classifiers will be demonstrated and finally CULP and CULM will be compared along all the classical approaches and the similar works around classification using complex networks.

As the first experiment, different link predictors are used in CULP to compare the performance of each one on the datasets. For this experiments the predictor $\lambda$ is one of the CN, AA, RA and CS which are respectively defined in Eqs. (2), (3), (4), (12).

The parameters used in CULM are $k$ ($1 \leq k \leq 35$), $\lambda$ (the link predictor which is Common Neighbors, Resource Allocation, Adamic Adar or Compatibility Score), the vector similarity function $s$ and $\alpha$ ($0.5 \leq \alpha \leq 1$). For each link predictor and each dataset, the parameters are tuned. This tuning is done via a 10-Fold Cross Validation procedure. After finding the best parameters, 30 runs of 10-Fold Cross Validation is done that amount to total of 300 runs. Table 3 captures the results obtained by these settings.

In each cell of Table 3, the first number is the mean accuracy of the runs and the second number is the standard deviation of them. The number in the parentheses represent the best $k$ obtained for each cell and the bold cell are the best result obtained on a dataset.

As can be seen in Table 3, the Compatibility Score achieved the best results among the predictors, this is due to the fact that CS exclusively got the highest accuracy on 6 datasets of Glass, Libras, Balance, Pima, Yeast and RedWine. In the second place is the Resource Allocation Index that obtained the top accuracy for Zoo, Iris, Ecoli, Optical and Poker exclusively and achieved an identical best accuracy with Adamic-Adar Score on the Vowel dataset. The third best predictor is the Common Neighbor with 5 datasets of Hayes, Teaching, Sonar, Thyroid and Vehicle on top and finally Adamic-Adar for Wine, Image and Segment and the shared best results with RA for Vowel.

Analyzing the $k$s in this experiments, we can see that for 10 datasets of Zoo, Hayes, Iris, Teaching, Wine, Image, Thyroid, Libras, Vehicle and Poker the best $k$ is identical for each predictor on a dataset; in Balance and Pima however; the $k$s are noticeably different with Common Neighbor having the highest $k$ in both of them. In the rest of the datasets the choice of $k$ among different predictors are at most different by 1 (for Yeast it is 2).

### 6.2. CULM analysis

As the next experiment, the CULM algorithm is run on each of the datasets. The parameter $\alpha$ is tuned over the set {0.6, 0.7, 0.8, 0.9, 1}. All the values below 0.6 for $\alpha$ is not used to keep the results and comparisons fair (as stated before, any value below 0.5 for $\alpha$ zeros the effect of CULP predictors also experimentally the same holds for $\alpha = 0.5$), this way we are sure that the link predictors is not completely overshadowed by the low level classifier. Other parameters of the algorithm and the tuning is done as before and again each cell is the result of 300 runs.

For a low level classifier to accompany the link predictors in CULM, three different algorithms have been chosen and used. These low level classifiers are *LDA* (Linear Discriminant Analysis), *CART* (Classification And Regression Trees) and multi-class *SVM* (Support Vector Machine) with RBF kernel.

Table 4 captures the results of this experiments. The first column is the best results for each of the datasets using CULP (Table 3); the next three columns are the results of CULM with respectively LDA, CART and SVM as $\phi$ and in each of the cells in these column the numbers in parentheses represent the $k$ and $\alpha$ used in runs. The last column in this table represents the accuracy gain achieved by using CULM instead of CULP. Each of the numbers in this column is obtained by comparing the best result obtained by CULM with the best result obtained by CULP for each dataset.

Looking at Table 4 it is clear that in the Thyroid dataset, using CULM achieved no change in the accuracy and in the datasets Iris and Optical the accuracy deteriorates; however, taking into account the other 17 datasets, CULM almost achieved a completely higher result.

CULM with SVM as its low level classifier achieved the best results on 6 datasets of Sonar, Thyroid, Libras, Balance, Vowel, RedWine and Poker exclusively and shares the best result on Thyroid with CULM-LDA and CULP. As the next best classifiers we have both CULM-CART and CULM-LDA with exclusively 5 best accuracy

**Table 3**
Accuracy of CULP on the dataset with different link predictors. The number in parentheses represent the $k$ used in runs.

| Dataset | CN | AA | RA | CS |
|---|---|---|---|---|
| Zoo | 95.567 ± 5.8 (2) | 96.567 ± 5.3 (2) | **96.833 ±5.4 (2)** | 96.767 ± 5.4 (2) |
| Hayes | **73.949 ± 12.0 (1)** | 73.718 ± 12.1 (1) | 73.718 ± 12.1 (1) | 73.667 ± 12.1 (1) |
| Iris | 98.467 ± 3.0 (11) | 98.467 ± 3.0 (11) | **98.489 ± 3.0 (11)** | 98.378 ± 3.2 (11) |
| Teaching | **63.756 ± 11.3 (1)** | 63.356 ± 11.1 (1) | 63.356 ± 11.1 (1) | 63.622 ± 11.2 (1) |
| Wine | 98.549 ± 2.8 (12) | **98.745 ± 2.6 (12)** | 98.725 ± 2.7 (12) | 98.137 ± 3.2 (12) |
| Sonar | **87.467 ± 7.4 (2)** | 87.250 ± 7.3 (3) | 86.900 ± 7.3 (3) | 87.100 ± 7.5 (3) |
| Image | 88.333 ± 6.7 (3) | **89.317 ± 6.3 (3)** | 89.175 ± 6.3 (3) | 89.063 ± 6.4 (3) |
| Glass | 71.857 ± 9.1 (3) | 73.540 ± 9.2 (3) | 73.397 ± 9.3 (2) | **74.048 ± 9.1 (2)** |
| Thyroid | **97.540 ± 3.1 (4)** | 97.413 ± 3.2 (4) | 97.413 ± 3.2 (4) | 97.333 ± 3.3 (4) |
| Ecoli | 86.798 ± 6.1 (9) | 87.010 ± 6.0 (8) | **87.141 ± 6.1 (9)** | 87.030 ± 6.0 (8) |
| Libras | 79.935 ± 6.5 (2) | 82.472 ± 6.3 (2) | 81.713 ± 6.4 (2) | **82.750 ± 6.2 (2)** |
| Balance | 93.753 ± 2.9 (6) | 96.446 ± 2.2 (2) | 96.446 ± 2.2 (2) | **96.780 ± 2.2 (2)** |
| Pima | 76.061 ± 4.5 (34) | 76.154 ± 4.4 (28) | 76.211 ± 4.4 (28) | **76.355 ± 4.3 (7)** |
| Vehicle | **73.611 ± 4.4 (5)** | 73.091 ± 4.7 (5) | 73.198 ± 4.7 (5) | 72.512 ± 4.7 (5) |
| Vowel | 97.603 ± 1.6 (3) | **98.242 ± 1.5 (2)** | **98.242 ± 1.5 (2)** | 97.886 ± 1.5 (2) |
| Yeast | 59.682 ± 3.9 (22) | 59.971 ± 3.7 (20) | 60.032 ± 3.6 (20) | **60.365 ± 3.8 (22)** |
| RedWine | 60.501 ± 3.9 (1) | 60.166 ± 3.9 (2) | 60.036 ± 3.9 (2) | **60.574 ± 3.8 (2)** |
| Segment | 96.333 ± 1.3 (3) | **96.535 ± 1.2 (4)** | 96.525 ± 1.2 (4) | 96.281 ± 1.3 (4) |
| Optical | 98.805 ± 0.4 (5) | 98.905 ± 0.4 (5) | **98.918 ± 0.4 (5)** | 98.851 ± 0.4 (4) |
| Poker | 58.518 ± 0.9 (32) | 58.604 ± 0.9 (32) | **58.625 ± 0.9 (32)** | 58.520 ± 0.9 (32) |

**Table 4**
Accuracy of CULM on the dataset with different link predictors. The first column is the best results obtained using CULP on each of the datasets. The number in parentheses represent the $k$ and $\alpha$ used in runs.

| Dataset | CULP | **CULM**-*LDA* | **CULM**-*CART* | **CULM**-*SVM* | Gain |
|---|---|---|---|---|---|
| Zoo | 96.833 ± 5.4 (RA,2) | 97.467 ± 5.3 (1,0.6) | **97.500 ± 5.0** (1,0.6) | 97.000 ± 5.9 (1,0.6) | +0.667 |
| Hayes | 73.949 ± 12.0 (CN,1) | 74.513 ± 11.6 (1,0.7) | **76.949 ± 11.1** (1,0.6) | 76.487 ± 11.1 (1,0.6) | +3.000 |
| Iris | **98.489 ± 3.0** (RA,11) | 98.467 ± 3.0 (11,0.7) | 98.467 ± 3.0 (11,0.7) | 98.467 ± 3.0 (11,0.7) | −0.022 |
| Teaching | 63.756 ± 11.3 (CN,1) | **65.667 ± 11.6** (1,0.6) | 64.200 ± 12.0 (1,0.6) | 65.622 ± 11.7 (1,0.6) | +1.911 |
| Wine | 98.745 ± 2.6 (AA,12) | **98.843 ± 2.9** (12,0.7) | 98.706 ± 2.7 (12,0.7) | 98.745 ± 2.6 (12,0.7) | +0.098 |
| Sonar | 87.467 ± 7.4 (CN,2) | 87.233 ± 7.2 (2,0.6) | 87.050 ± 7.4 (3,0.7) | **87.817 ± 7.3** (2,0.6) | +0.350 |
| Image | 89.317 ± 6.3 (AA,3) | **90.349 ± 6.2** (3,0.6) | 90.333 ± 6.0 (3,0.6) | 89.571 ± 6.3 (3,0.7) | +1.032 |
| Glass | 74.048 ± 9.1 (CS,2) | 74.095 ± 9.1 (2,0.6) | **74.952 ± 8.8** (2,0.6) | 74.365 ± 9.4 (2,0.6) | +0.904 |
| Thyroid | **97.540 ± 3.1** (CN,4) | **97.540 ± 3.1** (4,0.6) | 97.492 ± 3.1 (4,0.6) | **97.540 ± 3.1** (4,0.6) | 0 |
| Ecoli | 87.141 ± 6.1 (RA,9) | 87.475 ± 5.8 (8,0.6) | **87.495 ± 5.9** (8,0.6) | 87.293 ± 5.8 (9,0.6) | +0.354 |
| Libras | 82.750 ± 6.2 (CS,2) | 82.843 ± 6.0 (2,0.6) | 82.370 ± 5.8 (2,0.6) | **82.944 ± 5.9** (1,0.6) | +0.194 |
| Balance | 96.780 ± 2.2 (CS,2) | 97.016 ± 2.0 (2,0.6) | 96.694 ± 2.1 (2,0.7) | **97.946 ± 1.7** (2,0.6) | +1.166 |
| Pima | 76.355 ± 4.3 (CS,7) | **76.535 ± 4.5** (7,0.6) | 76.461 ± 4.5 (7,0.6) | 76.373 ± 4.6 (7,0.6) | +0.180 |
| Vehicle | 73.611 ± 4.4 (CN,5) | **74.829 ± 4.6** (5,0.6) | 73.897 ± 4.5 (5,0.6) | 74.167 ± 4.6 (5,0.6) | +1.218 |
| Vowel | 98.242 ± 1.5 (AA,2) | 98.461 ± 1.3 (2,0.9) | 98.508 ± 1.4 (2,0.9) | **98.620 ± 1.3** (2,0.8) | +0.378 |
| Yeast | **60.365 ± 3.8** (CS,22) | 60.360 ± 3.6 (20,0.6) | 60.288 ± 3.7 (20,0.6) | 60.113 ± 3.7 (20,1) | −0.005 |
| RedWine | 60.574 ± 3.8 (CS,2) | 64.170 ± 3.7 (1,0.6) | 63.453 ± 3.7 (1,0.6) | **64.447 ± 3.6** (1,0.6) | +3.873 |
| Segment | 96.535 ± 1.2 (AA,4) | 96.673 ± 1.3 (2,0.6) | **96.922 ± 1.2** (2,0.6) | 96.651 ± 1.3 (2,0.6) | +0.387 |
| Optical | **98.918 ± 0.4** (RA,5) | 98.905 ± 0.4 (4,0.9) | 98.890 ± 0.4 (4,0.9) | 98.890 ± 0.4 (4,0.9) | −0.013 |
| Poker | 58.625 ± 0.9 (RA,32) | 58.581 ± 0.9 (32,1) | 58.695 ± 0.9 (32,0.6) | **58.760 ± 0.9** (32,0.6) | +0.135 |

each (Zoo, Hayes, Glass, Ecoli and Segment for CULM-CART and Teaching, Wine, Image, Pima and Vehicle for CULM-LDA).

Datasets Hayes and RedWine achieved the highest accuracy gain (more than 3%) using CULM which is a noticeable boost. In the next level are datasets Teaching, Image Balance and Vehicle with more than 1% gain. In general, the collective amount of gain achieved using CULM is the average of 0.8% through all datasets which is another proof that CULM achieves a better results than CULP.

As for the parameter $k$, more robustness can be observed among different CULM classifiers than variations of CULP. Except for the datasets Sonar, Ecoli and Libras, the choice of $k$ in all variations of CULM are identical, also in these three datasets this parameter is different by at most 1 on each classifier.

The other parameter $\alpha$ in this experiments reveals interesting facts as well. Except for the CULM-SVM on Yeast data and CULM-LDA on Poker dataset, we can observe $\alpha < 1$ in all the experiments; this shows that using the low level features through the low level classifier did indeed help the classification accuracy. Saying this, we still need a more detailed analysis on the effect of $\alpha$ on the accuracy which is the main discussion of the next segment.

### 6.3. $\alpha$ analysis

To analyze the $\alpha$ parameter further, six datasets were chosen, each with a single configuration to run with different $\alpha$ values. The datasets are Zoo with $k = 1$, Hayes with $k = 1$, Iris with $k = 11$, Teaching with $k = 1$, Wine with $k = 12$ and Sonar with $k = 2$ and $\alpha \in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ in each experiment. The choices for $\alpha$ is to demonstrate the effect of zeroing the effect of predictors ($\alpha \leq 0.5$), zeroing the effect of low level classifier ($\alpha = 1$) or picking something in between.

The results of this experiment are depicted in the charts of Fig. 3. Each chart represents the experiments done on a dataset. These charts capture the accuracy of each of the 3 CULM classifiers for each value of $\alpha$. Red lines are demonstrating the accuracy of CULM-LDA, black lines are CULM-CART and the gray lines depict the results of CULM-SVM.

As stated before, any value below 0.5 for $\alpha$ zeros the effect of CULP predictors, we also noted that experimentally the same holds for $\alpha = 0.5$. This is evident by looking at the plots of Fig. 3 because in all datasets and classifiers the accuracies obtained for $\alpha = 0.4$ and $\alpha = 0.5$ are identical.

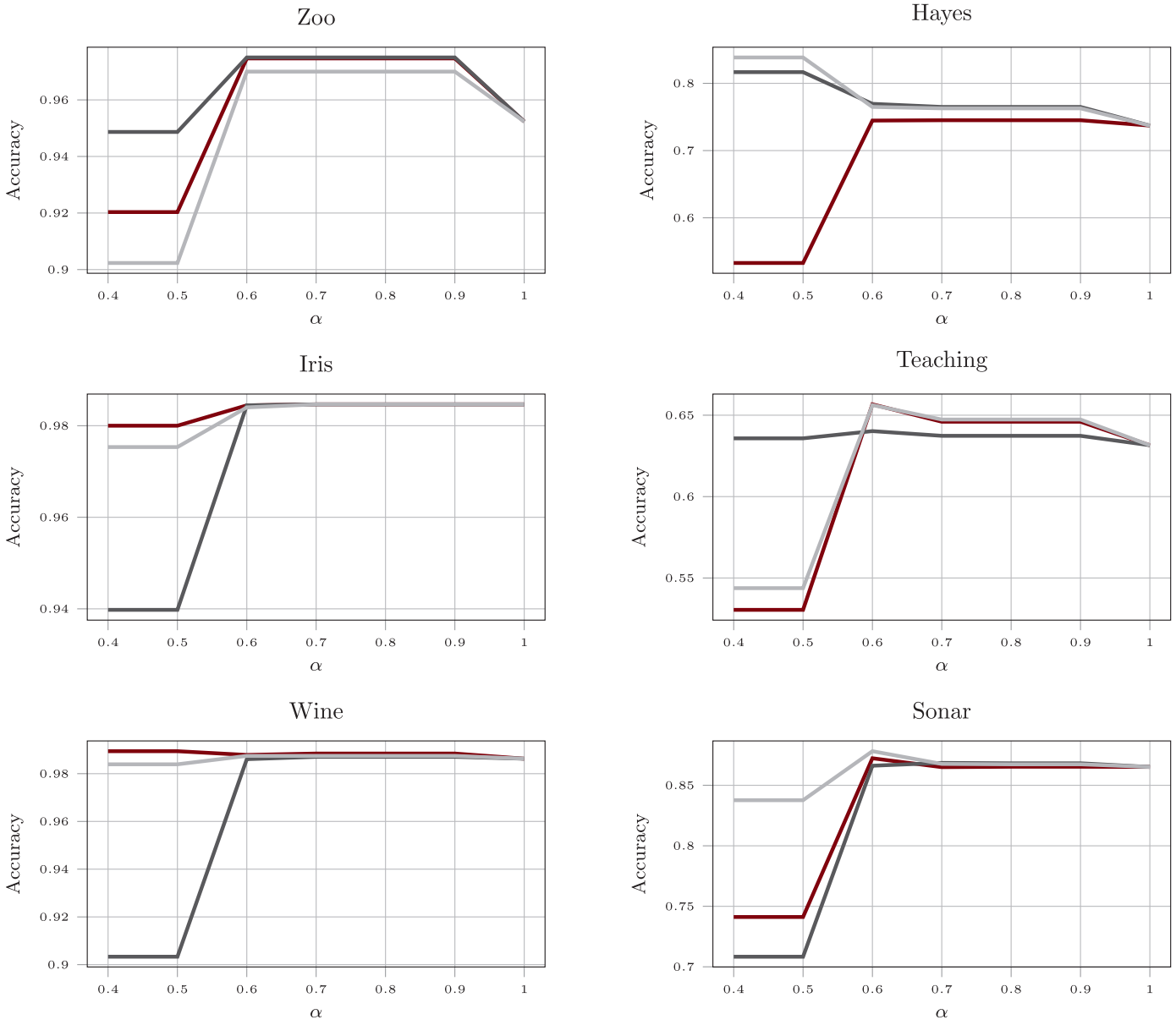


**Fig. 3.** Results of experimenting different values of $\alpha$ on 6 datasets. Each chart depicts accuracy on y axis and alpha on the x axis. Red lines are demonstrating the accuracy of CULM-LDA, black lines are CULM-CART and the gray lines depict the results of CULM-SVM.

As can be seen from the figure, for all classifiers of the datasets Zoo, Iris, Teaching and Sonar, using the predictors improved the accuracy of the low-level classifier; on the other hand, in all datasets zeroing the effect of the low-level classifier ($\alpha = 1$) had not helped (if not worsened) the accuracy of the prediction. The other notable detail in these plot is the plateau of accuracy for roughly the values of $\alpha$ between 0.6 and 0.9. This means that a less fine-grained set of values can be also used for tuning this parameter.

For the next experiment, the results of CULP and CULM is compared with 4 classical classifiers. These classifiers include *k*NN classifier, LDA, CART and multi-class SVM with RBF kernel.

### 6.4. Comparison to classical classifiers

The results of this experiment is captured in Table 5. In this table the first column represent the best result of CULP for each dataset, the second column is the best result of CULM for each dataset and the other 4 columns are the results obtained by the classical classifier. The number in the parentheses in the cells of first three column represent *k* and the bold cells are the best results obtained on the dataset.

Comparing the *k* values in the first 3 columns of Table 5, we can realize that except for Ecoli and Yeast, this parameter is smaller (or equal) for CULP and CULM than that of the *k*NN algorithm and in some cases like Wine and Balance this difference is quite high. This is due to the fact that the undirected version of *k* nearest neighbor is used to form up the LEG graph which consequently enables us to capture the similarity features with less neighbors.

It is evident from the results that CULP and CULM achieved superior results compared to the classical algorithms. CULP and CULM collectively obtained the best results on 13 datasets. The *k*NN and LDA algorithms achieved the highest results on 3 datasets each, SVM got the best results only on the Hayes dataset and CART is completely outperformed by the other algorithms on all datasets.

One thing that can be noted is the fact that CULM could obtain the best results on the datasets Hayes, Wine, Pima and Vehicle

**Table 5**
Accuracy of comparing CULP and CULM with 4 different classical classifiers. The number in the parentheses in the cells of first three column represent $k$ and the bold cells are the best results obtained on the dataset.

| Dataset | CULP | CULM | $k$NN | LDA | CART | SVM |
|---|---|---|---|---|---|---|
| Zoo | 96.833 ± 5.4 (2) | **97.500 ± 5.0 (1)** | 97.500 ± 12.7 (1) | 92.033 ± 8.2 | 94.867 ± 6.8 | 90.233 ± 10.0 |
| Hayes | 73.949 ± 12.0 (1) | 76.949 ± 11.1 (1) | 72.590 ± 14.6 (1) | 53.282 ± 13.0 | 81.667 ± 9.5 | **85.103 ± 8.4** |
| Iris | **98.489 ± 3.0 (11)** | 98.467 ± 3.0 (11) | 97.222 ± 5.4 (17) | 98.000 ± 3.5 | 94.556 ± 5.6 | 97.533 ± 3.8 |
| Teaching | 63.756 ± 11.3 (1) | **65.667 ± 11.6 (1)** | 62.511 ± 12.7 (1) | 53.044 ± 13.2 | 64.333 ± 11.4 | 54.378 ± 12.7 |
| Wine | 98.745 ± 2.6 (12) | 98.843 ± 2.9 (12) | 97.000 ± 5.6 (24) | **98.941 ± 2.4** | 90.294 ± 6.9 | 98.392 ± 3.0 |
| Sonar | 87.467 ± 7.4 (2) | **87.817 ± 7.3 (2)** | 86.383 ± 8.0 (1) | 74.117 ± 9.1 | 70.850 ± 9.7 | 83.767 ± 7.8 |
| Image | 89.317 ± 6.3 (3) | **90.349 ± 6.2 (3)** | 85.667 ± 8.3 (4) | 89.635 ± 6.1 | 88.222 ± 6.8 | 87.317 ± 6.4 |
| Glass | 74.048 ± 9.1 (2) | **74.952 ± 8.8 (2)** | 72.683 ± 10.4 (1) | 62.381 ± 10.0 | 66.698 ± 9.1 | 70.190 ± 9.8 |
| Thyroid | **97.540 ± 3.1 (4)** | **97.540 ± 3.1 (4)** | 96.206 ± 5.8 (1) | 91.397 ± 5.7 | 93.857 ± 5.4 | 95.921 ± 4.0 |
| Ecoli | 87.141 ± 6.1 (9) | **87.495 ± 5.9 (8)** | 86.909 ± 6.3 (7) | 86.869 ± 5.6 | 79.515 ± 6.8 | 86.828 ± 6.0 |
| Libras | 82.750 ± 6.2 (2) | 82.944 ± 5.9 (1) | **85.880 ± 8.0 (1)** | 64.620 ± 8.4 | 68.713 ± 8.2 | 80.306 ± 6.6 |
| Balance | 96.780 ± 2.2 (2) | **97.946 ± 1.7 (2)** | 90.140 ± 5.5 (15) | 86.747 ± 3.9 | 81.306 ± 5.9 | 90.489 ± 3.6 |
| Pima | 76.355 ± 4.3 (7) | 76.535 ± 4.5 (7) | 74.171 ± 4.8 (9) | **77.320 ± 4.3** | 70.289 ± 5.1 | 76.013 ± 4.4 |
| Vehicle | 73.611 ± 4.4 (5) | 74.829 ± 4.6 (5) | 72.206 ± 5.2 (6) | **78.052 ± 4.3** | 71.282 ± 4.9 | 76.675 ± 4.8 |
| Vowel | 98.242 ± 1.5 (2) | 98.620 ± 1.3 (2) | **98.983 ± 2.3 (1)** | 59.556 ± 4.7 | 81.192 ± 4.2 | 94.852 ± 2.3 |
| Yeast | **60.365 ± 3.8 (20)** | 60.360 ± 3.6 (20) | 59.586 ± 3.8 (19) | 58.923 ± 3.8 | 51.205 ± 4.0 | 60.124 ± 3.7 |
| RedWine | 60.574 ± 3.8 (2) | 64.447 ± 3.6 (1) | **64.662 ± 3.8 (1)** | 59.172 ± 3.9 | 63.390 ± 3.8 | 62.637 ± 3.7 |
| Segment | 96.535 ± 1.2 (4) | **96.922 ± 1.2 (2)** | 95.829 ± 1.8 (1) | 91.446 ± 1.9 | 95.459 ± 1.4 | 93.825 ± 1.6 |
| Optical | **98.918 ± 0.4 (5)** | 98.905 ± 0.4 (4) | 98.823 ± 0.5 (3) | 95.278 ± 0.8 | 90.532 ± 1.3 | 98.681 ± 0.5 |
| Poker | 58.625 ± 0.9 (32) | **58.760 ± 0.9 (32)** | 58.517 ± 1.0 (34) | 49.952 ± 0.9 | 48.948 ± 1.7 | 58.617 ± 0.5 |

**Table 6**
Accuracies of CULP and CULM, classical classifiers, HLCRW [2] and PgRkNN [9] algorithms.

| Dataset | CULP | CULM | Classical | HLCRW | PgRkNN |
|---|---|---|---|---|---|
| Zoo | 96.833 ± 5.4 | 97.500 ± 5.0 | 97.500 ± 12.7 | 97.00 ± 0.1 | **99.03 ± 2.9** |
| Hayes | 73.949 ± 12.0 | 76.949 ± 11.1 | **85.103 ± 8.4** | 61.70 ± 2.3 | 73.09 ± 11.7 |
| Iris | **98.489 ± 3.0** | 98.467 ± 3.0 | 98.000 ± 3.5 | 98.00 ± 0.6 | 97.20 ± 3.7 |
| Teaching | 63.756 ± 11.3 | **65.667 ± 11.6** | 64.333 ± 11.4 | 65.30 ± 2.0 | 62.08 ± 13.4 |
| Wine | 98.745 ± 2.6 | 98.843 ± 2.9 | **98.941 ± 2.4** | 87.10 ± 1.6 | 93.95 ± 5.3 |
| Sonar | 87.467 ± 7.4 | **87.817 ± 7.3** | 86.383 ± 8.0 | 81.79 ± 7.8 | 82.00 ± 7.5 |
| Image | 89.317 ± 6.3 | **90.349 ± 6.2** | 89.635 ± 6.1 | 75.60 ± 0.8 | 86.13 ± 7.2 |
| Glass | 74.048 ± 9.1 | **74.952 ± 8.8** | 72.683 ± 10.4 | 72.80 ± 1.1 | 71.75 ± 7.9 |
| Thyroid | 97.540 ± 3.1 | 97.540 ± 3.1 | 96.206 ± 5.8 | **97.57 ± 3.0** | 97.55 ± 3.0 |
| Ecoli | 87.141 ± 6.1 | **87.495 ± 5.9** | 86.909 ± 6.3 | 85.50 ± 0.6 | 85.11 ± 5.4 |
| Libras | 82.750 ± 6.2 | 82.944 ± 5.9 | 85.880 ± 8.0 | 85.00 ± 0.8 | **87.16 ± 9.8** |
| Balance | 96.780 ± 2.2 | **97.946 ± 1.7** | 90.489 ± 3.6 | 97.20 ± 0.6 | 90.86 ± 3.4 |
| Pima | 76.355 ± 4.3 | 76.535 ± 4.5 | **77.320 ± 4.3** | 75.54 ± 4.6 | 74.85 ± 4.9 |
| Vehicle | 73.611 ± 4.4 | 74.829 ± 4.6 | **78.052 ± 4.3** | 67.70 ± 0.6 | 70.26 ± 4.1 |
| Vowel | 98.242 ± 1.5 | 98.620 ± 1.3 | **98.983 ± 2.3** | 97.50 ± 0.3 | 98.49 ± 1.2 |
| Yeast | **60.365 ± 3.8** | 60.360 ± 3.6 | 60.124 ± 3.7 | 57.20 ± 0.5 | 56.50 ± 3.6 |
| RedWine | 60.574 ± 3.8 | 64.447 ± 3.6 | 64.662 ± 3.8 | 61.60 ± 0.5 | **66.68 ± 3.5** |
| Segment | 96.535 ± 1.2 | **96.922 ± 1.2** | 95.829 ± 1.8 | 93.20 ± 0.2 | 95.63 ± 1.5 |
| Optical | 98.918 ± 0.4 | 98.905 ± 0.4 | 98.823 ± 0.5 | 95.09 ± 2.1 | **98.94 ± 0.4** |
| Poker | 58.625 ± 0.9 | **58.760 ± 0.9** | 58.617 ± 0.5 | 55.42 ± 0.9 | 53.78 ± 0.8 |

with $\alpha \leq 0.5$ but as stated before we decided to forgo these values to give a fair comparison; however, in general we can state that CULM can outperform or achieve the same result of any classical classification algorithms given the right configuration for the $\alpha$ parameter.

### 6.5. Complete comparison

As the final experiment of this paper, a complete comparison is done to analyze the results of CULM, CULP, the classical algorithms and two of the similar works that use complex network representation of the data to classify the unlabeled instances. These two classifiers which were discussed in the related work sections are PgRkNN [9] and HLCRW [2] (short for High Level data Classification using Random Walk)

The results of PgRkNN and HLCRW algorithms on datasets which were already provided in their papers are used here without a change, for other cases we implemented and run both of them completely by the details provided in those papers.

Table 6 captures these results along with the summaries of Tables 3–5. For each of the rows in this table the bold cell is the best result for classifying the instances of the dataset through all of the algorithms. The best result for each of the cases where CULP/CULM obtained the higher average accuracy, is tested for significance against the second best accuracy using the Welch's $t$-test with confidence level of 0.95. In this test, the null hypothesis is that the averages are the same and the alternative hypothesis is that they are different. Except for the Teaching dataset which the bold and underlined values are not significantly different all the other bold values in CULP and CULM columns are superior.

In the first glance at Table 6 it can be realized that CULM is the leader with 8 best results on the datasets among different algorithms. These datasets include Teaching, Sonar, Image, Glass, Ecoli, Balance, Segment and Poker.The next best algorithm in case of the best results is the Classical group with 5 dataset of Hayes, Wine, Pima, Vehicle and Vowel in lead. As the third algorithm we have PgRkNN with datasets Zoo, Libras, RedWine and Optical. The one before last is CULP with Iris and Yeast on top and finally HLCRW with only Thyroid with the best result.

In order to give a more thorough view on the ranking of the algorithm of Tables 6 and 7 is formed. In this table the best

**Table 7**
Rankings of the algorithms of Table 6.

| Dataset | CULP | CULM | Classical | HLCRW | PgRkNN |
|---|---|---|---|---|---|
| Zoo | 5 | 2 | 2 | 4 | 1 |
| Hayes | 3 | 2 | 1 | 5 | 4 |
| Iris | 1 | 2 | 4 | 3 | 5 |
| Teaching | 4 | 1 | 3 | 2 | 5 |
| Wine | 3 | 2 | 1 | 5 | 4 |
| Sonar | 2 | 1 | 3 | 5 | 4 |
| Image | 3 | 1 | 2 | 5 | 4 |
| Glass | 2 | 1 | 4 | 3 | 5 |
| Thyroid | 3 | 3 | 5 | 1 | 2 |
| Ecoli | 2 | 1 | 3 | 4 | 5 |
| Libras | 5 | 4 | 2 | 3 | 1 |
| Balance | 3 | 1 | 5 | 2 | 4 |
| Pima | 3 | 2 | 1 | 4 | 5 |
| Vehicle | 3 | 2 | 1 | 5 | 4 |
| Vowel | 4 | 2 | 1 | 5 | 3 |
| Yeast | 1 | 2 | 3 | 4 | 5 |
| RedWine | 5 | 3 | 2 | 4 | 1 |
| Segment | 2 | 1 | 3 | 5 | 4 |
| Optical | 2 | 3 | 4 | 5 | 1 |
| Poker | 2 | 1 | 3 | 4 | 5 |
| Average | 2.925 | 1.9 | 2.675 | 3.9 | 3.6 |

result on a dataset gets 1 and the worst gets a 5. In case of ties the algorithms get the same value and when computing the average rankings, the ties effect their averages as the mean of their respective ranks (if 2 algorithms are both ranked 3, they sum up as 3.5 to compute the average rank).

As can be seen in Table 7, CULM has the best rank of 1.9 which is far better than the second ranked Classical algorithms (rank 2.675). The third rank belongs to CULP with 2.925 and after that comes PgRkNN and HLCRW with 3.55 and 3.95, respectively. These are evidence that CULP and CULM are highly accurate classifiers and competitive with classical and similar works.

## 7. Conclusion

In this work, we proposed a novel way to look at the problem of classification using a link prediction scope. Our proposed memory efficient graph data structure LEG enabled the use of any link predictor to assist the classification procedure and captured not only the unlabeled and labeled data, but also the classes in a unified manner.

Our proposed algorithm CULP can be used with any link predictor to derive the class of the unlabeled data. In this work Common Neighbors, Adamic-Adar Index and resource allocation were used along with our own local link predictor called Compatibility Score as the predictors for CULP. Our algorithm demonstrated superiority to similar algorithms which use graph representations to classify a data point and our Compatibility Score was also one of the best predictors in our experiments.

We also extend CULP by a weighted majority vote with weights proportional to the probabilities of the predictions. CULM is the name of our extension which not only uses multiple predictors but it also exploits the low level features of the data as well.

Our experiments on both CULM and CULP showed high accuracy on 20 different datasets and superiority on all the classical approaches and similar graph based methods.

## 8. Future works

There are a lot to be done with all the proposed methods and algorithms elaborated in this paper. We are going to test our Compatibility Score on graph datasets and test its accuracy on explicit link prediction problems. Another idea in our agenda is testing both CULP and CULM algorithms with other link prediction methods, possibly more complex ones such as random walk or matrix factorization to analyze any further improvement. Finally, a stacking approach to find the weights of CULM is under construction which hopefully be discussed in another work.

## Declaration of competing interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome. We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed.

We further confirm that the order of authors listed in the manuscript has been approved by all of us. We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property.

We understand that the Corresponding Author is the sole contact for the Editorial process (including Editorial Manager and direct communications with the office). She is responsible for communicating with the other authors about progress, submissions of revisions and final approval of proofs.

## References

[1] K.P. Murphy, Machine Learning: A Probabilistic Perspective, The MIT Press, 2012.
[2] T.H. Cupertino, M.G. Carneiro, Q. Zheng, J. Zhang, L. Zhao, A scheme for high level data classification using random walk and network measures, Expert Syst. Appl. 92 (2018) 289–303.
[3] T.C. Silva, L. Zhao, Network-based high level data classification, IEEE Trans. Neural Netw. Learn. Syst. 23 (6) (2012) 954–970.
[4] T.F. Covões, Z. Liang, Low and high level classification using stacking, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 2525–2532.
[5] M.G. Carneiro, L. Zhao, High level classification totally based on complex networks, in: Proceedings of the 11th Brazilian Congress on Computational Intelligence (BRICS-CCI & CBIC), IEEE, 2013, pp. 507–514.
[6] J.R. Bertini Jr, L. Zhao, R. Motta, A. de Andrade Lopes, A nonparametric classification method based on k-associated graphs, Inf. Sci. 181 (24) (2011) 5435–5456.
[7] F.A. Neto, L. Zhao, High level data classification based on network entropy, in: Neural Networks (IJCNN), The 2013 International Joint Conference on, IEEE, 2013, pp. 1–5.
[8] T.H. Cupertino, T.C. Silva, L. Zhao, Classification of multiple observation sets via network modularity, Neural Comput. Appl. 23 (7–8) (2013) 1923–1929.
[9] M.G. Carneiro, L. Zhao, Organizational data classification based on the importance concept of complex networks, IEEE Trans. Neural Netw. Learn. Syst. (2017).
[10] R. Angelova, G. Weikum, Graph-based text classification: learn from your neighbors, in: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2006, pp. 485–492.
[11] J. Bai, S. Xiang, C. Pan, A graph-based classification method for hyperspectral images, IEEE Trans. Geosci. Remote Sens. 51 (2) (2013) 803–817.
[12] G. Camps-Valls, T.V.B. Marsheva, D. Zhou, Semi-supervised graph-based hyperspectral image classification, IEEE Trans. Geosci. Remote Sens. 45 (10) (2007) 3044–3054.
[13] V. Martínez, F. Berzal, J.-C. Cubero, A survey of link prediction in complex networks, ACM Comput. Surv. (CSUR) 49 (4) (2017) 69.
[14] D. Liben-Nowell, J. Kleinberg, The link-prediction problem for social networks, J. Assoc. Inf. Sci. Technol. 58 (7) (2007) 1019–1031.
[15] L.A. Adamic, E. Adar, Friends and neighbors on the web, Soc. Netw. 25 (3) (2003) 211–230.
[16] T. Zhou, L. Lü, Y.-C. Zhang, Predicting missing links via local information, Eur. Phys. J. B 71 (4) (2009) 623–630.
[17] M. Belkin, P. Niyogi, Laplacian Eigenmaps for dimensionality reduction and data representation, Neural Comput. 15 (6) (2003) 1373–1396.
[18] J. Chen, H.-r. Fang, Y. Saad, Fast approximate KNN graph construction for high dimensional data via recursive Lanczos bisection, J. Mach. Learn. Res. 10 (Sep) (2009) 1989–2012.
[19] D. Dua, C. Graff, UCI Machine Learning Repository, Irvine, CA, 2017. University of California, School of Information and Computer Science. http://archive.ics.uci.edu/ml.

**Seyed Amin Fadaee** born in 1994 is a Persian Msc. Artificial Intelligence student at Amirkabir University of Technology with major in complex network analysis. His B.Sc. degree was in software engineering, graduated in 2017. He is currently working as a data science researcher in Amirkabir University of Technology.

**Maryam Amir Haeri** received the B.Sc. degree in Software Engineering, and the M.Sc. degree in Information Technology from the Sharif University of Technology, Tehran, Iran, in 2007 and 2009, respectively. She also received the Ph.D. degree in Artificial Intelligence from the Amirkabir University of Technology, Tehran, Iran, in 2014. Since September 2015, she is assistant professor in the Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran. Her research interests include Big Data Analytics, Complex Networks Analysis, Soft Computing, Machine Learning and Data Mining.