Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models

Citation for published version:
Rózemberczki, B & Sarkar, R 2020, Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management.* ACM Association for Computing Machinery, New York, NY, USA, pp. 1325–1334, 29th ACM International Conference on Information and Knowledge Management, Omline Conference, 19/10/20. https://doi.org/10.1145/3340531.3411866

Digital Object Identifier (DOI):
10.1145/3340531.3411866

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Proceedings of the 29th ACM International Conference on Information & Knowledge Management

Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models

Benedek Rozemberczki
The University of Edinburgh
Edinburgh, United Kingdom
benedek.rozemberczki@ed.ac.uk

Rik Sarkar
The University of Edinburgh
Edinburgh, United Kingdom
rsarkar@inf.ed.ac.uk

ABSTRACT

In this paper, we propose a flexible notion of characteristic functions defined on graph vertices to describe the distribution of vertex features at multiple scales. We introduce *FEATHER*, a computationally efficient algorithm to calculate a specific variant of these characteristic functions where the probability weights of the characteristic function are defined as the transition probabilities of random walks. We argue that features extracted by this procedure are useful for node level machine learning tasks. We discuss the pooling of these node representations, resulting in compact descriptors of graphs that can serve as features for graph classification algorithms. We analytically prove that *FEATHER* describes isomorphic graphs with the same representation and exhibits robustness to data corruption. Using the node feature characteristic functions we define parametric models where evaluation points of the functions are learned parameters of supervised classifiers. Experiments on real world large datasets show that our proposed algorithm creates high quality representations, performs transfer learning efficiently, exhibits robustness to hyperparameter changes and scales linearly with the input size.

ACM Reference Format:
Benedek Rozemberczki and Rik Sarkar. 2020. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3340531.3411866

1 INTRODUCTION

Recent works in network mining have focused on characterizing node neighbourhoods. Features of a neighbourhood serve as valuable inputs to downstream machine learning tasks such as node classification, link prediction and community detection [3, 15, 18, 29, 44]. In social networks, the importance of neighbourhood features arises from the property of homophily (correlation of network connections with similarity of attributes), and social neighbours have been shown to influence habits and attributes of individuals [31]. Attributes of a neighbourhood is found to be important in other types of networks as well. Network mining methods have used aggregate features from several degrees of neighbourhoods for network analysis and embedding [18, 44, 45, 47].

Neighbourhood features can be complex to interpret. Network datasets can incorporate multiple attributes, with varied distributions that influence the characteristics of a node and the network. Attributes such as income, wealth or number of page accesses can have an unbounded domain, with unknown distributions. Simple linear aggregates [18, 44, 45, 47] such as the mean values do not represent this diverse information.

We use characteristic functions [7] as a rigorous but versatile way of utilising diverse neighborhood information. A unique characteristic function always exists irrespective of the nature of the distribution, and characteristic functions can be meaningfully composed across multiple nodes and even multiple attributes. These features let us represent and compare different neighborhoods in a unified framework.

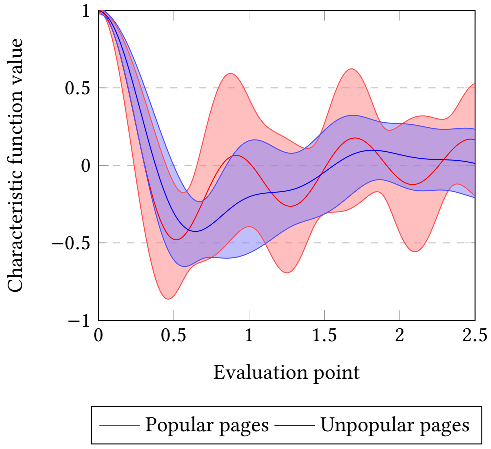

Figure 1: The real part of class dependent mean characteristic functions with standard deviations around the mean for the log transformed degree on the Wikipedia Crocodiles dataset.

Figure 1 shows the distribution of node level characteristic function values on the Wikipedia Crocodiles web graph [33]. In this dataset nodes are webpages which have two types of labels – popular and unpopular. With log transformed degree centrality as the vertex attribute, we conditioned the distributions on the class memberships. We plotted the mean of the distribution at each evaluation point with the standard deviation around the mean. One can easily observe that the value of the characteristic function is

CIKM '20, October 19–23, 2020, Virtual Event, Ireland
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3411866

discriminative with respect to the class membership of nodes. Our experimental results about node classification in Subsection 4.2 validates this observation about characteristic functions for the Wikipedia dataset and various other social networks.

Present work. We propose complex valued characteristic functions [7] for representation of neighbourhood feature distributions. Characteristic functions are analogues of Fourier Transforms defined for probability distributions. We show that these continuous functions can be evaluated suitably at discrete points to obtain effective characterisation of neighborhoods and describe an approach to learn the appropriate evaluation points for a given task.

The correlation of attributes are known to decrease with the decrease in tie strength, and with increasing distance between nodes [6]. We use a random-walk based tie strength definition, where tie strength at the scale r between a source and target node pair is the probability of an r length random walk from the source node ending at the target. We define the *r-scale random walk weighted characteristic function* as the characteristic function weighted by these tie strengths. We propose *FEATHER* an algorithm to efficiently evaluate this function for multiple features on a graph.

We theoretically prove that graphs which are isomorphic have the same pooled characteristic function when the mean is used for pooling node characteristic functions. We argue that the *FEATHER* algorithm can be interpreted as the forward pass of a parametric statistical model (e.g. logistic regression or a feed-forward neural network). Exploiting this we define the r-scale random walk weighted characteristic function based softmax regression and graph neural network models (respectively named *FEATHER-L* and *FEATHER-N*).

We evaluate *FEATHER* model variants on two machine learning tasks – node and graph classification. Using data from various real world social networks (Facebook, Deezer, Twitch) and web graphs (Wikipedia, GitHub), we compare the performance of *FEATHER* with graph neural networks, neighbourhood preserving and attributed node embedding techniques. Our experiments illustrate that *FEATHER* outperforms comparable unsupervised methods by as much as 4.6% on node labeling and 12.0% on graph classification tasks in terms of test AUC score. The proposed procedures show competitive transfer learning capabilities on social networks and the supervised *FEATHER* variants show a considerable advantage over the unsupervised model, especially when the number of evaluation points is limited. Runtime experiments establish that *FEATHER* scales linearly with the input size.

Main contributions. To summarize, our paper makes the following contributions:

(1) We introduce a generalization of characteristic functions to node neighbourhoods, where the probability weights of the characteristic function are defined by tie strength.
(2) We discuss a specific instance of these functions – the r-scale random walk weighted characteristic function. We propose *FEATHER*, an algorithm that calculates these characteristic functions efficiently to create Euclidean node embeddings.
(3) We demonstrate that this function can be applied simultaneously to multiple features.
(4) We show that the r-scale random walk weighted characteristic function calculated by *FEATHER* can serve as the building block for an end-to-end differentiable parametric classifier.
(5) We experimentally assess the behaviour of *FEATHER* on real world node and graph classification tasks.

The remainder of this work has the following structure. In Section 2 we overview the relevant literature on node embedding techniques, graph kernels and neural networks. We introduce characteristic functions defined on graph vertices in Section 3 and discuss using them as building blocks in parametric statistical models. We empirically evaluate *FEATHER* on various node and graph classification tasks, transfer learning problems, and test its sensitivity to hyperparameter changes in Section 4. The paper concludes with Section 5 where we discuss our main findings and point out directions for future work. The newly introduced node classification datasets and a Python reference implementation of *FEATHER* is available at https://github.com/benedekrozemberczki/FEATHER.

2 RELATED WORK

Characteristic functions have previously been used in relation to heat diffusion wavelets [10], which defined the functions for uniform ties strengths and restricted features types.

Node embedding techniques map nodes of a graph into Euclidean spaces where the similarity of vertices is approximately preserved – each node has a vector representation. Various forms of embeddings have been studied recently, *Neighbourhood preserving* node embeddings are learned by explicitly [3, 27, 32] or implicitly decomposing [29, 30, 39] a proximity matrix of the graph. *Attributed node embedding* techniques [25, 44–47] augment the neighbourhood information with generic node attributes (e.g. the user's age in a social network) and nodes sharing metadata are closer in the learned embedding space. *Structural embeddings* [2, 15, 18] create vector representations of nodes which retain the similarity of structural roles and equivalences of nodes. The non-supervised *FEATHER* algorithm which we propose can be seen as a node embedding technique. We create a mapping of nodes to the Euclidean space, simply by evaluating the characteristic function for metadata based generic, neighbourhood and structural node attributes. With the appropriate tie strength definition we are able to hybridize all three types of information with our embedding.

Whole graph embedding and *statistical graph fingerprinting* techniques map graphs into Euclidean spaces where graphs with similar structures and subpatterns are located in close proximity – each graph obtains a vector representation. Whole graph embedding procedures [4, 26] achieve this by decomposing graph – structural feature matrices to learn an embedding. Statistical graph fingerprinting techniques [8, 12, 40, 42] extract information from the graph Laplacian eigenvalues, or using the graph scattering transform without learning. Node level *FEATHER* representations can be pooled by permutation invariant functions to output condensed graph fingerprints which is in principle similar to statistical graph fingerprinting. These statistical fingerprints are related to *graph kernels* as the pooled characteristic functions can serve as inputs for appropriate kernel functions. This way the similarity of graphs is not compared based on the presence of sparsely appearing common random walks [14], cyclic patterns [19] or subtree patterns [37], but via the kernel defined on pairs of dense pooled graph characteristic function representations.

There is also a parallel between *FEATHER* and the forward pass of *graph neural network layers* [17, 22]. During the *FEATHER* function evaluation using the tie strength weights and vertex features we create multi-scale descriptors of the feature distributions which are parameterized by the evaluation points. This can be seen as the forward pass of a multi-scale graph neural network [1, 23] which is able to describe vertex features at multiple scales. Using this we essentially define end-to-end differentiable parametric statistical models where the modulation of evaluation points (the relaxation of fixed evaluation points) can help with the downstream learning task at hand. Compared to traditional graph neural networks [1, 5, 17, 22, 23, 43], which only calculate the first moments of the node feature distributions, *FEATHER* models give summaries of node feature distributions with trainable characteristic function evaluation points.

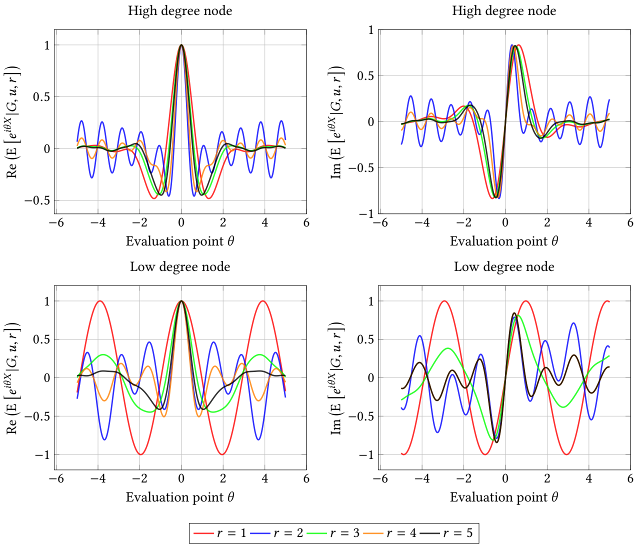

Figure 2: The real and imaginary parts of the r-scale random walk weighted characteristic function of the log transformed degree for a low degree and high degree node from the Twitch England graph.

3 CHARACTERISTIC FUNCTIONS ON GRAPHS

In this section we introduce the idea of characteristic functions defined on attributed graphs. Specifically, we discuss the idea of describing node feature distributions in a neighbourhood with characteristic functions. We propose a specific instance of these functions, the *r-scale random walk weighted characteristic function* and we describe an algorithm to calculate this function for all nodes in linear time. We prove the robustness of these functions and how they represent isomorphic graphs when node level functions are pooled. Finally, we discuss how characteristic functions can serve as building blocks for parametric statistical models.

3.1 Node feature distribution characterization

We assume that we have an attributed and undirected graph $G = (V, E)$. Nodes of G have a feature described by the random variable X, specifically defined as the feature vector $\mathbf{x} \in \mathbb{R}^{|V|}$, where $\mathbf{x}_v$ is the feature value for node $v \in V$. We are interested in describing the distribution of this feature in the neighbourhood of $u \in V$. The characteristic function of X for source node u at characteristic function evaluation point $\theta \in \mathbb{R}$ is the function defined by Equation (1) where i denotes the imaginary unit.

$$\mathrm{E}\left[e^{i\theta X}|G,u\right] = \sum_{w\in V} P(w|u)\cdot e^{i\theta \mathbf{x}_w} \tag{1}$$

In Equation (1) the *affiliation probability* $P(w|u)$ describes the strength of the relationship between the source node u and the target node $w \in V$. We would like to emphasize that the source node u and the target nodes do not have to be connected directly and that $\sum_{w\in V} P(w|u) = 1$ holds $\forall u \in V$. We use Euler's identity to obtain the real and imaginary part of the function described by Equation (1) which are respectively defined by Equations (2) and (3).

$$\mathrm{Re}\left(\mathrm{E}\left[e^{i\theta X}|G,u\right]\right) = \sum_{w\in V} P(w|u)\cos(\theta\mathbf{x}_w) \tag{2}$$

$$\mathrm{Im}\left(\mathrm{E}\left[e^{i\theta X}|G,u\right]\right) = \sum_{w\in V} P(w|u)\sin(\theta\mathbf{x}_w) \tag{3}$$

The real and imaginary parts of the characteristic function are respectively weighted sums of cosine and sine waves where the weight of an individual wave is $P(w|u)$, the evaluation point θ is equivalent to time, and $\mathbf{x}_w$ describes the angular frequency.

3.1.1 The r-scale random walk weighted characteristic function. So far we have not specified how the affiliation probability $P(w|u)$ between the source u and target w is parametrized. Now we will introduce a parametrization which uses random walk transition probabilities. The sequence of nodes in a random walk on G is denoted by $\{v_j, v_{j+1}, \ldots, v_{j+r}\}$.

Let us assume that the neighbourhood of u at scale r consists of nodes that can be reached by a random walk in r steps from source node u. We are interested in describing the distribution of the feature in the neighbourhood of $u \in V$ at scale r with the real and imaginary parts of the characteristic function – these are respectively defined by Equations (4) and (5).

$$\mathrm{Re}\left(\mathrm{E}\left[e^{i\theta X}|G,u,r\right]\right) = \sum_{w\in V} P(v_{j+r} = w|v_j = u)\cos(\theta\mathbf{x}_w) \tag{4}$$

$$\mathrm{Im}\left(\mathrm{E}\left[e^{i\theta X}|G,u,r\right]\right) = \sum_{w\in V} P(v_{j+r} = w|v_j = u)\sin(\theta\mathbf{x}_w) \tag{5}$$

In Equations (4) and (5), $P(v_{j+r} = w|v_j = u) = P(w|u)$ is the probability of a random walk starting from source node u, hitting the target node w in the r^{th} step. The adjacency matrix of G is denoted by $\mathbf{A}$ and the weighted diagonal degree matrix is $\mathbf{D}$. The normalized adjacency matrix is defined as $\widehat{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$. We can exploit the fact that, for a source-target node pair (u, w) and a scale r, we can express $P(v_{j+r} = w|v_j = u)$ with the r^{th} power of the normalized adjacency matrix. Using $\widehat{\mathbf{A}}^r_{u,w} = P(v_{j+r} = w|v_j = u)$, we get Equations (6) and (7).

$$\mathrm{Re}\left(\mathrm{E}\left[e^{i\theta X}|G,u,r\right]\right)=\sum_{w\in V}\widehat{\mathbf{A}}^{r}_{u,w}\cos(\theta\mathbf{x}_w) \tag{6}$$

$$\mathrm{Im}\left(\mathrm{E}\left[e^{i\theta X}|G,u,r\right]\right)=\sum_{w\in V}\widehat{\mathbf{A}}^{r}_{u,w}\sin(\theta\mathbf{x}_w) \tag{7}$$

Figure 2 shows the real and imaginary part of the *r-scale random walk weighted characteristic function* of the log transformed degree for a low and high degree node in the Twitch England network [33]. A few important properties of the function are visible; (i) the real part is an even function while the imaginary part is odd, (ii) the range of both parts is in the [-1,1] interval, (iii) nodes with different structural roles have different characteristic functions.

3.1.2 Efficient calculation of the r-scale random walk weighted characteristic function. Up to this point we have only discussed the characteristic function at scale r for a single $u \in V$. However, we might want to characterize every node with respect to a feature in the graph in an efficient way. Moreover, we do not want to evaluate each node characteristic function on the whole domain. Because of this we will sample d points from the domain and evaluate the function at these points which are described by the *evaluation point vector* $\Theta \in \mathbb{R}^d$. We define the r-scale random walk weighted characteristic function of the whole graph as the complex matrix valued function denoted as $\mathcal{CF}(G,X,\Theta,r)\to\mathbb{C}^{|V|\times d}$. The real and imaginary parts of this complex matrix valued function are described by the matrix valued functions in Equations (8) and (9) respectively.

$$\mathrm{Re}(\mathcal{CF}(G,X,\Theta,r))=\widehat{\mathbf{A}}^{r}\cdot\cos(\mathbf{x}\otimes\Theta) \tag{8}$$

$$\mathrm{Im}(\mathcal{CF}(G,X,\Theta,r))=\widehat{\mathbf{A}}^{r}\cdot\sin(\mathbf{x}\otimes\Theta) \tag{9}$$

These matrices describe the feature distributions around nodes if two rows are similar it, implies that the corresponding nodes have similar distributions of the feature around them at scale r. This representation can be seen as a *node embedding*, which characterizes the nodes in terms of the local feature distribution. Calculating the r-scale random walk weighted characteristic function for the whole graph has a time complexity of $O(|E|\cdot d\cdot r)$ and memory complexity of $O(|V|\cdot d)$.

3.1.3 Characterizing multiple features for all nodes. Up to this point, we have assumed that the nodes have a single feature, described by the feature vector $\mathbf{x}\in\mathbb{R}^{|V|}$. Now we will consider the more general case when we have a set of k node feature vectors. In a social network these vectors might describe the age, income, and other generic real valued properties of the users. This set of vertex features is defined by $\mathcal{X}=\{\mathbf{x}^1,\dots,\mathbf{x}^k\}$.

We now consider the design of an efficient sparsity aware algorithm which can calculate the r scale random walk weighted characteristic function for each node and feature. We named this procedure *FEATHER*, and it is summarized by Algorithm 1. It evaluates the characteristic functions for a graph for each feature $\mathbf{x}\in\mathcal{X}$ at all scales up to r. The connectivity of the graph is described by the normalized adjacency matrix $\widehat{\mathbf{A}}$. For each feature vector $\mathbf{x}^i,\ i\in 1,\dots,k$ at scale r we have a corresponding characteristic function evaluation vector $\Theta^{i,r}\in\mathbb{R}^d$. For simplicity we assume that we evaluate the characteristic functions at the same number of points.

Let us look at the mechanics of Algorithm 1. First, we initialize the real and imaginary parts of the embeddings denoted by $\mathbf{Z}_{Re}$ and $\mathbf{Z}_{Im}$ respectively (lines 1 and 2). We iterate over the k different node features (line 3) and the scales up to r (line 4). When we consider the first scale (line 6) we calculate the outer product of the feature being considered and the corresponding evaluation point vector – this results in $\mathbf{H}$ (line 7). We elementwise take the sine and cosine of this matrix (lines 8 and 9). For each scale we calculate the real and imaginary parts of the graph characteristic function evaluations ($\mathbf{H}_{Re}$ and $\mathbf{H}_{Im}$) – we use the normalized adjacency matrix to define the probability weights (lines 10 and 11). We append these matrices to the real and imaginary part of the embeddings (lines 13 and 14). When the characteristic function of each feature is evaluated at every scale we concatenate the real and imaginary part of the embeddings (line 17) and we return this embedding (line 18).

```
Data: $\widehat{\mathbf{A}}$ – Normalized adjacency matrix.
      $\mathcal{X}=\{\mathbf{x}^1,\dots,\mathbf{x}^k\}$ – Set of node feature vectors.
      $\widetilde{\Theta}=\{\Theta^{1,1},\dots,\Theta^{1,r},\Theta^{2,1},\dots,\Theta^{k,r}\}$ – Set of evaluation point
vectors.
      $r$ – Scale of empirical graph characteristic function.
Result: Node embedding matrix $\mathbf{Z}$.
1  $\mathbf{Z}_{Re}\leftarrow$ Initialize Real Features()
2  $\mathbf{Z}_{Im}\leftarrow$ Initialize Imaginary Features()
3  for $i$ in $1:k$ do
4      for $j$ in $1:r$ do
5          for $l$ in $1:j$ do
6              if $l=1$ then
7                  $\mathbf{H}\leftarrow\mathbf{x}^i\otimes\Theta^{i,j}$
8                  $\mathbf{H}_{Re}\leftarrow\cos(\mathbf{H})$
9                  $\mathbf{H}_{Im}\leftarrow\sin(\mathbf{H})$
10             $\mathbf{H}_{Re}\leftarrow\widehat{\mathbf{A}}\mathbf{H}_{Re}$
11             $\mathbf{H}_{Im}\leftarrow\widehat{\mathbf{A}}\mathbf{H}_{Im}$
12         end
13         $\mathbf{Z}_{Re}\leftarrow[\mathbf{Z}_{Re}\mid\mathbf{H}_{Re}]$
14         $\mathbf{Z}_{Im}\leftarrow[\mathbf{Z}_{Im}\mid\mathbf{H}_{Im}]$
15     end
16 end
17 $\mathbf{Z}\leftarrow[\mathbf{Z}_{Im}\mid\mathbf{Z}_{Re}]$
18 Output $\mathbf{Z}$.
```

Algorithm 1: Efficient r-scale random walk weighted characteristic function calculation for multiple node features.

Calculating the outer product (line 7) $\mathbf{H}$ takes $O(|V|\cdot d)$ memory and time. The probability weighting (lines 10 and 11) is an operation which requires $O(|V|\cdot d)$ memory and $O(|E|\cdot d)$ time. We do this for each feature at each scale with a separate evaluation point vector. This means that altogether calculating the r scale graph characteristic function for each feature has a time complexity of $O((|E|+|V|)\cdot d\cdot r^2\cdot k)$ and the memory complexity of storing the embedding is $O(|V|\cdot d\cdot r\cdot k)$.

3.2 Theoretical properties

We focus on two theoretical aspects of the r-scale random weighted characteristic function which have practical implications: robustness and how it represents isomorphic graphs.

Remark 1. *Let us consider a graph G, the feature X and its corrupted variant X' represented by the vectors $\mathbf{x}$ and $\mathbf{x}'$. The corrupted* *feature vector only differs from* $\boldsymbol{x}$ *at a single node* $w \in V$ *where* $\boldsymbol{x}'_w = \boldsymbol{x}_w \pm \varepsilon$ *for any* $\varepsilon \in \mathbb{R}$*. The absolute changes in the real and imaginary parts of the* r*-scale random walk weighted characteristic function for any* $u \in V$ *and* $\theta \in \mathbb{R}$ *satisfy that:*

$$\underbrace{\left|\mathrm{Re}\left(\mathrm{E}\left[e^{i\theta X}|G,u,r\right]\right) - \mathrm{Re}\left(\mathrm{E}\left[e^{i\theta X'}|G,u,r\right]\right)\right|}_{\Delta\mathrm{Re}} \leq 2 \cdot \widehat{\mathbf{A}}^r_{u,w}$$

$$\underbrace{\left|\mathrm{Im}\left(\mathrm{E}\left[e^{i\theta X}|G,u,r\right]\right) - \mathrm{Im}\left(\mathrm{E}\left[e^{i\theta X'}|G,u,r\right]\right)\right|}_{\Delta\mathrm{Im}} \leq 2 \cdot \widehat{\mathbf{A}}^r_{u,w}.$$

Proof. We know that the absolute difference in the real and imaginary part of the characteristic function is bounded by the maxima of such differences:

$$|\Delta\mathrm{Re}| \leq \max|\Delta\mathrm{Re}| \quad \text{and} \quad |\Delta\mathrm{Im}| \leq \max|\Delta\mathrm{Im}|.$$

We will prove the bound for the real part, the proof for the imaginary one follows similarly. Let us substitute the difference of the characteristic functions in the right hand side of the bound:

$$|\Delta\mathrm{Re}| \leq \max\left|\sum_{v\in V}\widehat{\mathbf{A}}^r_{u,v}\cos(\theta\mathbf{x}_v) - \sum_{v\in V}\widehat{\mathbf{A}}^r_{u,v}\cos(\theta\mathbf{x}'_v)\right|.$$

We exploit that $x_v = x'_v,\ \forall v \in V \setminus \{w\}$ so we can rewrite the difference of sums because $\cos(\theta\mathbf{x}_v) - \cos(\theta\mathbf{x}'_v) = 0,\ \forall v \in V \setminus \{w\}$.

$$|\Delta\mathrm{Re}| \leq \max\left|\sum_{v\in V\setminus\{w\}}\left[\widehat{\mathbf{A}}^r_{u,v}\underbrace{\left(\cos(\theta\mathbf{x}_v) - \cos(\theta\mathbf{x}'_v)\right)}_{0}\right] + \widehat{\mathbf{A}}^r_{u,w}\left(\cos(\theta\mathbf{x}_w) - \cos(\theta\mathbf{x}'_w)\right)\right|$$

The maximal absolute difference between two cosine functions is 2 so our proof is complete which means that the effect of corrupted features on the r-scale random walk weighted characteristic function values is bounded by the tie strength regardless the extent of data corruption.

$$|\Delta\mathrm{Re}| \leq \widehat{\mathbf{A}}^r_{u,w} \cdot \underbrace{\max\left|\cos(\theta\mathbf{x}_w) - \cos(\theta\mathbf{x}'_w)\right|}_{2}$$

□

Definition 1. *The real and imaginary part of the* ***mean pooled*** r*-scale random walk weighted characteristic function are defined as* $\sum\limits_{u\in V}\sum\limits_{w\in V}\widehat{\boldsymbol{A}}^r_{u,w}\cos(\theta\boldsymbol{x}_w)/|V|$ *and* $\sum\limits_{u\in V}\sum\limits_{w\in V}\widehat{\boldsymbol{A}}^r_{u,w}\sin(\theta\boldsymbol{x}_w)/|V|$.

The functions described by Definition 1 allow for the characterization and comparison of whole graphs based on structural properties. Moreover, these descriptors can serve as features for graph level machine learning algorithms.

Remark 2. *Given two isomorphic graphs* G, G' *and the respective degree vectors* $\boldsymbol{x}, \boldsymbol{x}'$ *the mean pooled* r*-scale random walk weighted degree characteristic functions are the same.*

Proof. Let us denote the normalized adjacency matrices of G and G' as $\widehat{\mathbf{A}}$ and $\widehat{\mathbf{A}}'$. Because G and G' are isomorphic there is a $\mathbf{P}$ permutation matrix for which it holds that $\widehat{\mathbf{A}} = \mathbf{P}\widehat{\mathbf{A}}'\mathbf{P}^{-1}$. Using the same permutation matrix we get that $\mathbf{x} = \mathbf{P}\mathbf{x}'$. Using Definition 1 and the previous two equations it follows that the real and imaginary parts of pooled characteristic functions satisfy that

$$\sum_{u\in V}\sum_{w\in V}\widehat{\mathbf{A}}^r_{u,w}\cos(\theta\mathbf{x}_w)/|V| = \sum_{u\in V}\sum_{w\in V}(\mathbf{P}\widehat{\mathbf{A}}\mathbf{P}^{-1})^r_{u,w}\cos(\theta\cdot(\mathbf{P}\mathbf{x})_w)/|V|$$

$$\sum_{u\in V}\sum_{w\in V}\widehat{\mathbf{A}}^r_{u,w}\sin(\theta\mathbf{x}_w)/|V| = \sum_{u\in V}\sum_{w\in V}(\mathbf{P}\widehat{\mathbf{A}}\mathbf{P}^{-1})^r_{u,w}\sin(\theta\cdot(\mathbf{P}\mathbf{x})_w)/|V|.$$

□

3.3 Parametric characteristic functions

Our discussion postulated that the evaluation points of the r-scale random walk characteristic function are predetermined. However, we can define parametric models where these evaluation points are learned in a semi-supervised fashion to make the evaluation points selected the most discriminative with regards to a downstream classification task. The process which we describe in Algorithm 1 can be interpreted as the forward pass of a graph neural network which uses the normalized adjacency matrix and node features as input. This way the evaluation points and the weights of the classification model could be learned jointly in an end-to-end fashion.

3.3.1 Softmax parametric model. Now we define the classifiers with learned evaluation points, let $\mathbf{Y}$ be the $|V| \times C$ one-hot encoded matrix of node labels, where C is the number of node classes. Let us assume that $\mathbf{Z}$ was calculated by using Algorithm 1 in a forward pass with a trainable $\widetilde{\Theta}$. The classification weights of the softmax characteristic function classifier are defined by the trainable weight matrix $\beta \in \mathbb{R}^{(2\cdot k\cdot d\cdot r)\times C}$. The class label distribution matrix of nodes outputted by the softmax characteristic function model is defined by Equation (10) where the softmax function is applied row-wise. We reference this supervised softmax model as *FEATHER-L*.

$$\widehat{\mathbf{Y}} = \mathrm{softmax}(\mathbf{Z}\cdot\beta) \tag{10}$$

3.3.2 Neural parametric model. We introduce the forward pass of a neural characteristic function model with a single hidden layer of feed forward neurons. The trainable input weight matrix is $\beta_0 \in \mathbb{R}^{(2\cdot k\cdot d\cdot r)\times h}$ and the output weight matrix is $\beta_1 \in \mathbb{R}^{h\times C}$, where h is the number of neurons in the hidden layer. The class label distribution matrix of nodes output by the neural model is defined by Equation (11), where $\sigma(\cdot)$ is an activation function applied element-wise (in our experiments it is a ReLU). We refer to neural models with this architecture as *FEATHER-N*.

$$\widehat{\mathbf{Y}} = \mathrm{softmax}(\sigma(\mathbf{Z}\cdot\beta_0)\cdot\beta_1) \tag{11}$$

3.3.3 Optimizing the parametric models. The log-loss of the *FEATHER-N* and *FEATHER-L* models being minimized is defined by Equation (12) where $U \subseteq V$ is the set of labeled training nodes.

$$\mathcal{L} = -\sum_{u\in U}\sum_{c=1}^{C}\mathbf{Y}_{u,c}\cdot\log(\widehat{\mathbf{Y}}_{u,c}) \tag{12}$$

This loss is minimized with a variant of gradient descent to find the optimal values of β (respectively β_0 and β_1) and $\widetilde{\Theta}$. The softmax model has $O(k\cdot r\cdot d\cdot C)$ while the neural model has $O(k\cdot r\cdot d\cdot h + C\cdot h)$ free trainable parameters, As a comparison, generating the representations upstream with *FEATHER* and learning a logistic regression has $O(k\cdot r\cdot d\cdot C)$ trainable parameters.

4 EXPERIMENTAL EVALUATION

In this section, we overview the datasets used to quantify representation quality. We demonstrate how node and graph features distilled with *FEATHER* can be used to solve node and graph classification tasks. Furthermore, we highlight the transfer learning capabilities, scalability and robustness of our method.

4.1 Datasets

We briefly discuss the real world datasets and their descriptive statistics which we use to evaluate the node and graph features extracted with our proposed methods.

Table 1: Statistics of social networks used for the evaluation of node classification algorithms, sensitivity analysis, and transfer learning.

Dataset	Nodes	Density	Clustering Coefficient	Diameter	Unique Features	Classes
Wiki Croco	11,631	0.003	0.026	11	13,183	2
FB Page-Page	22,470	0.001	0.232	15	4,714	4
LastFM ASIA	7,624	0.001	0.179	15	7,842	18
Deezer EUR	28,281	0.002	0.096	21	31,240	2
Twitch DE	9,498	0.003	0.047	7	3,169	2
Twitch EN	7,126	0.002	0.042	10	3,169	2
Twitch ES	4,648	0.006	0.084	9	3,169	2
Twitch PT	1,912	0.017	0.131	7	3,169	2
Twitch RU	4,385	0.004	0.049	9	3,169	2
Twitch TW	2,772	0.017	0.120	7	3,169	2

4.1.1 Node level datasets. We used various publicly available, and self-collected social network and webgraph datasets to evaluate the quality of node features extracted with *FEATHER*. The descriptive statistics of these datasets are summarized in Table 1. These graphs are heterogeneous with respect to size, density, and number of features, and they allow for binary and multi-class node classification.

- **Wikipedia Crocodiles [35]:** A webgraph of Wikipedia articles about crocodiles where each node is a page and edges are mutual links between edges. Attributes represent nouns appearing in the articles and the binary classification task on the dataset is deciding whether a page is popular or not.
- **Twitch Social Networks [33]:** Social networks of gamers from the streaming service Twitch. Features describe the history of games played and the task is to predict whether a gamer streams adult content. The country specific graphs share the same node features which means that we can perform transfer learning with these datasets.
- **Facebook Page-Page [33]:** A webgraph of verified Facebook pages which liked each other. Features were extracted from page descriptions and the classification target is the page category.
- **LastFM Asia:** The LastFM Asia graph is a social network of users from Asian (e.g. Philippines, Malaysia, Singapore) countries which we collected. Nodes represent users of the music streaming service LastFM and links among them are friendships. We collected these datasets in March 2020 via the use of the API. The classification task related to these two datasets is to predict the home country of a user given the social network and artists liked by the user.
- **Deezer Europe:** A social network of European Deezer users which we collected from the public API in March 2020. Nodes represent users and links are mutual follower relationships among users. The related classification task is the prediction of gender using the friendship graph and the artists liked.

4.1.2 Graph level datasets. We utilized a range of publicly available non-attributed, social graph datasets to assess the quality of graph level features distilled via our procedure. Summary statistics, enlisted in Table 2, demonstrate that these datasets have a large number of small graphs with varying size, density and diameter.

- **Reddit Threads [35]**: A collection of Reddit thread and non-thread based discussion graphs. The related task is to correctly classify graphs according the thread – non-thread categorization.
- **Twitch Egos [35]** The ego-networks of Twitch users who participated in the partnership program. The classification task entails the identification of gamers who only play with a single game.
- **GitHub Repos [35]:** Social networks of developers who starred machine learning and web design repositories. The target is the type of the repository itself.
- **Deezer Egos [35]:** A small collection of ego-networks for European Deezer users. The related task involves the prediction of the ego node's gender.

Table 2: Statistics of graph datasets used for the evaluation of graph classification algorithms.

		Nodes		Density		Diameter	
Dataset	Graphs	Min	Max	Min	Max	Min	Max
Reddit Threads	203,088	11	97	0.021	0.382	2	27
Twitch Egos	127,094	14	52	0.038	0.967	1	2
GitHub Repos	12,725	10	957	0.003	0.561	2	18
Deezer Egos	9,629	11	363	0.015	0.909	2	2

4.2 Node classification

The node classification performance of *FEATHER* variants is compared to neighbourhood based, structural and attributed node embedding techniques. We also studied the performance in contrast to various competitive graph neural network architectures.

4.2.1 Experimental settings. We report mean micro averaged test AUC scores with standard errors calculated from 10 seeded splits with a 20%/80% train-test split ratio in Table 3.

The unsupervised neighbourhood based [3, 29, 30, 32, 36, 39], structural [2] and attributed node [25, 33, 44–47] embeddings were created by the *Karate Club* [35] software package and used the default hyperparameter settings of the 1.0 release. This ensure that the number of free parameters used to represent the nodes by the upstream unsupervised models is the same. We used the publicly available official Python implementation of *Node2Vec* [15] with the default settings and the *In-Out* and *Return* hyperparameters were fine tuned with 5-fold cross validation within the training set. The downstream classifier was a logistic regression which used the default hyperparameter settings of *Scikit-Learn* [28] with the *SAGA* optimizer [9].

Table 3: Mean micro-averaged AUC values on the test set with standard errors on the node level datasets calculated from 10 seed train-test splits. Black bold numbers denote the best performing unsupervised model, while blue ones denote the best supervised one.

	Wikipedia Crocodiles	Facebook Page-Page	LastFM Asia	Deezer Europe
DeepWalk [29]	.820 ± .001	.880 ± .001	.918 ± .001	.520 ± .001
LINE [39]	.856 ± .001	.956 ± .001	.949 ± .001	.543 ± .001
Walklets [30]	.872 ± .001	.975 ± .001	**.950 ± .001**	.547 ± .001
HOPE [27]	.855 ± .001	.903 ± .002	.922 ± .001	.539 ± .001
NetMF [32]	.859 ± .001	.946 ± .001	.943 ± .001	.538 ± .001
Node2Vec [15]	.850 ± .001	.974 ± .001	.944 ± .001	.556 ± .001
Diff2Vec [36]	.812 ± .001	.867 ± .001	.907 ± .001	.521 ± .001
GraRep [3]	.871 ± .001	.951 ± .001	.926 ± .001	.547 ± .001
Role2Vec [2]	.801 ± .001	.911 ± .001	.924 ± .001	.534 ± .001
GEMSEC [34]	.858 ± .001	.933 ± .001	**.951 ± .001**	.544 ± .001
ASNE [25]	.853 ± .001	.933 ± .001	.910 ± .001	.528 ± .001
BANE [45]	.534 ± .001	.866 ± .001	.610 ± .001	.521 ± .001
TENE [46]	.893 ± .001	.874 ± .001	.855 ± .002	.639 ± .002
TADW [44]	.901 ± .001	.849 ± .001	.851 ± .001	.644 ± .001
SINE [47]	.895 ± .001	.975 ± .001	.944 ± .001	.618 ± .001
GCN [22]	.924 ± .001	**.984 ± .001**	.962 ± .001	.632 ± .001
GAT [41]	.917 ± .002	**.984 ± .001**	.956 ± .001	.611 ± .002
GraphSAGE [17]	.916 ± .001	**.984 ± .001**	.955 ± .001	.618 ± .001
ClusterGCN [5]	.922 ± .001	.977 ± .001	.944 ± .002	.594 ± .002
APPNP [23]	.900 ± .001	**.986 ± .001**	**.968 ± .001**	.667 ± .001
MixHop [1]	.928 ± .001	.976 ± .001	.956 ± .001	**.682 ± .001**
SGConv [43]	.889 ± .001	.966 ± .001	.957 ± .001	.647 ± .001
FEATHER	**.943 ± .001**	**.981 ± .001**	**.954 ± .001**	**.651 ± .001**
FEATHER-L	**.944 ± .002**	**.984 ± .001**	.960 ± .001	.656 ± .001
FEATHER-N	**.947 ± .001**	**.987 ± .001**	**.970 ± .001**	.673 ± .001

Supervised baselines were implemented with the *PyTorch Geometric* framework [11] and as a pre-processing step, the dimensionality of vertex features was reduced to be 128 by the *Scikit-Learn* implementation of Truncated SVD [16]. Each supervised model considers neighbours from 2 hop neighbourhoods except *APPNP* [23] which used a teleport probability of 0.2 and 10 personalized PageRank approximation iterations. Models were trained with the Adam optimizer [21] with a learning rate of 0.01 for 200 epochs. The input layers of the models had 32 filters and between the final and input layer we used a 0.5 dropout rate [38] during training time. The *ClusterGCN* [5] models decomposed the graph with the *METIS* [20] algorithm before training – the number of clusters equaled the number of classes.

The *FEATHER* model variants used a combination of neighbourhood based, structural and generic vertex attributes as input besides the graph itself. Specifically we used:

- **Neighbourhood features:** We used Truncated SVD to extract 32 dimensional node features from the normalized adjacency matrix.
- **Structural features:** The log transformed degree and the clustering coefficient are used as structural vertex features.
- **Generic node features:** We reduced the dimensionality of generic vertex features with Truncated SVD to be 32 dimensional for each network.

The unupservised *FEATHER* model used 16 evaluation points per feature, which were initialized uniformly in the $[0, 5]$ domain, and a scale of $r = 2$. We used a logistic regression downstream classifier. The characteristic function evaluation points of the supervised *FEATHER-L* and *FEATHER-N* models were initialized similarly. Models were trained by the Adam optimizer with a learning rate of 0.001, for 50 epochs and the neural model had 32 neurons in the hidden layer.

4.2.2 Node classification performance. Our results in Table 3 demonstrate that the unsupervised *FEATHER* algorithm outperforms the proximity preserving, structural and attributed node embedding techniques. This predictive performance advantage varies between 0.4% and 4.6% in terms of micro averaged test AUC score. On the Wikipedia, Facebook and LastFM datasets, the performance difference is significant at an $\alpha = 1\%$ significance level. On these three datasets the best supervised *FEATHER* variant marginally outperforms *graph neural networks* between 0.1% and 2.1% in terms of test AUC. However, this improvement of classification is only significant on the Wikipedia dataset at $\alpha = 1\%$.

4.3 Graph classification

The graph classification performance of unsupervised and supervised *FEATHER* models is compared to that of implicit matrix factorization, spectral fingerprinting and graph neural network models.

4.3.1 Experimental settings. We report average test AUC values with standard errors on binary classification tasks calculated from 10 seeded splits with a 20%/80% train-test split ratio in Table 4.

The unsupervised implicit matrix factorization [4, 26] and spectral fingerprinting [8, 12, 40, 42] representations were produced by the *Karate Club* framework with the standard hyperparameter settings of the 1.0 release. The downstream graph classifier was a logistic regression model implemented in *Scikit-Learn*, with the standard hyperparameters and the *SAGA* [9] optimizer.

Supervised models used the one-hot encoded degree, clustering coefficient and eccentricity as node features. Each method was trained by minimizing log-loss with the Adam optimizer [21] using a learning rate of 0.01 for 10 epochs with a batch size of 32. We used two consecutive graph convolutional [22] layers with 32 and 16 filters and ReLu activations. In the case of mean and maximum pooling the output of the second convolutional layer was pooled and fed to a fully connected layer. We do the same with Sort Pooling [48] by keeping 5 nodes and flattening the output of the pooling layer to create graph representations. In the case of Top-K pooling [13] and SAG Pooling [24] we pool the nodes after each convolutional layer with a pooling ratio of 0.5 and output graph representations with a final max pooling layer. The output of advanced pooling layers [13, 24, 48] was fed to a fully connected layer. The output of the final layers was transformed by the softmax function.

We used the unsupervised *FEATHER* model to create graph descriptors. We pooled the node features extracted for each characteristic function evaluation point with a permutation invariant aggregation function such as the mean, maximum and minimum. Node level representations only used the log transformed degree as a feature. We set $r = 5$, $d = 25$, and initialized the characteristic function evaluation points in the $[0, 5]$ interval uniformly. Using these descriptors we utilized logistic regression as a classifier with the settings used with other unsupervised methods.

4.3.2 Graph classification performance. Our results demonstrate that our proposed pooled characteristic function based classification method outperforms both supervised and unsupervised graph classification methods on the Reddit Threads, Twitch Egos and Github Repos datasets. The performance advantage of *FEATHER* varies between 1.1% and 12.0% in terms of AUC, which is a significant peformance gain on all three of these datasets at an $\alpha = 1\%$ significance level. On the Deezer Egos dataset the disadvantage of our method is not significant, but specific supervised and unsupervised procedures have a somewhat better predictive performance in terms of test AUC. We also have evidence that the mean pooling of the node level characteristic functions provides superior peformance on most datasets considered.

Table 4: Mean AUC values with standard errors on the graph datasets calculated from 10 seed train-test splits. Bold numbers denote the model with the best performance.

	Reddit Threads	Twitch Egos	GitHub Repos	Deezer Egos
GL2Vec [4]	.754 ± .001	.670 ± .001	.532 ± .002	.500 ± .001
Graph2Vec [26]	.808 ± .001	.698 ± .001	.563 ± .002	.510 ± .001
SF [8]	.819 ± .001	.642 ± .001	.535 ± .001	.503 ± .001
NetLSD [40]	.817 ± .001	.630 ± .001	.614 ± .002	.525 ± .001
FGSD [42]	.822 ± .001	.699 ± .001	.650 ± .002	**.528 ± .001**
Geo-Scatter [12]	.800 ± .001	.695 ± .001	.532 ± .001	.524 ± .001
Mean Pool	.801 ± .002	.708 ± .001	.599 ± .003	.503 ± .001
Max Pool	.805 ± .001	.713 ± .001	.612 ± .013	.515 ± .001
Sort Pool [48]	.807 ± .001	.712 ± .001	.614 ± .010	**.528 ± .001**
Top K Pool [13]	.807 ± .001	.706 ± .002	.634 ± .001	.520 ± .003
SAG Pool [24]	.804 ± .001	.705 ± .002	.620 ± .001	.518 ± .003
FEATHER MIN	**.834 ± .001**	**.719 ± .001**	.694 ± .001	.518 ± .001
FEATHER MAX	**.831 ± .001**	**.718 ± .001**	.689 ± .001	.521 ± .002
FEATHER AVE	.823 ± .001	**.719 ± .001**	**.728 ± .002**	**.526 ± .001**

4.4 Sensitivity analysis

We investigated how the representation quality changes when the most important hyperparameters of the r-scale random walk weighted characteristic function are manipulated. Precisely, we looked at the scale of the r-scale random walk weighted characteristic function and the number of evaluation points.

We use the Facebook Page-Page dataset, with the standard (20% /80%) split and input the log transformed degree as a vertex feature. Figure 3 plots the average test AUC against the manipulated hyperparameter calculated from 10 seeded splits. The models were trained with the hyperparameter settings discussed in Subsection 4.2. We chose a scale of 5 when the number of evaluation points was modulated and used 25 evaluation points when the scale was manipulated. The evaluation points were initialized uniformly in the $[0, 5]$ domain.

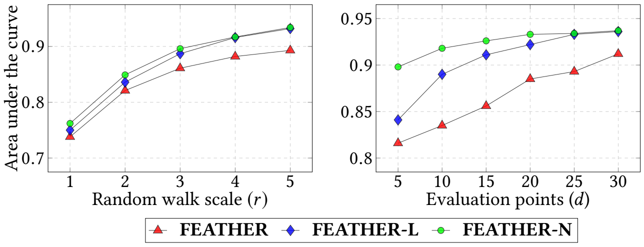

Figure 3: Mean AUC values on the Facebook page-page test set (10 seeded splits) achieved by FEATHER model variants as a function of random walk scale and characteristic function evaluation point count.

4.4.1 Scale of the characteristic function. First, we observe that including information from higher order neighbourhoods is valuable for the classification task. Second, the marginal performance increase is decreasing with the increase of the scale. Finally, when we only consider the first hop of neighbours we observe little performance difference between the unsupervised and supervised model variants. When higher order neighbourhoods are considered the supervised models have a considerable advantage.

4.4.2 Number of evaluation points. Increasing the number of characteristic function evaluation points increases the performance on the downstream predictive task. Supervised models are more efficient when the number of characteristic function evaluation points was low. The neural model is efficient in terms of the evaluation points needed for a good predictive performance. It is evident that the marginal predictive performance gains of the supervised models are decreasing with the number of evaluation points.

4.5 Transfer learning

Using the Twitch datasets, we demonstrate that the r-scale random walk weighted characteristic function features are robust and can be easily utilized in a transfer learning setup. We support evidence that the supervised parametric models also work in such scenarios. Figure 4 shows the transfer learning results for German, Spanish and Portuguese users, where the predictive performance is measured by average AUC values based on 10 experiments.

Each model was trained with nodes of a fully labeled source graph and evaluated on the nodes of the target graph. This transfer learning experiment requires that graphs share the target variable (abusing the Twitch platform), and that the characteristic function is calculated for the same set of node features. All models utilized the log transformed degree centrality of nodes as a shared and cheap-to-calculate structural feature. We used a scale of $r = 5$ and $d = 25$ characteristic function evaluation points for each *FEATHER* model variant. Models were fitted with the hyperparameter settings described in Subsection 4.2.

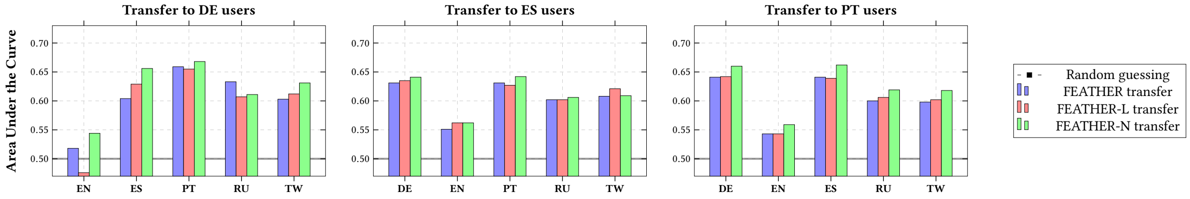

Figure 4: Transfer learning performance of FEATHER variants on the Twitch Germany, Spain and Portugal datasets as target graphs. The transfer performance was evaluated by mean AUC values calculated from 10 seeded experimental repetitions.

Firstly, our results presented on Figure 4 support that even the characteristic function of a single structural feature is sufficient for transfer learning as we are able to outperform the random guessing of labels in most transfer scenarios. Secondly, we see that the neural model has a predictive performance advantage over the unsupervised *FEATHER* and the shallow *FEATHER-L* model. Specifically, for the Portuguese users this advantage varies between 2.1 and 3.3% in terms of average AUC value. Finally, transfer from the English users seems to be poor to the other datasets. Which implies that the abuse of the platform is associated with different structural features in that case.

4.6 Runtime performances

We evaluated the runtime needed for calculating the proposed r-scale random walk weighted characteristic function. Using a synthetic ErdÅŚs-RÃĺnyi graph with 2^{12}nodes and 2^4 edges per node, we measured the runtime of Algorithm 1 for various values of the size of the graph, number of features and characteristic function evaluation points. Figure 5 shows the mean logarithm of the runtimes against the manipulated input parameter based on 10 experimental repetitions.

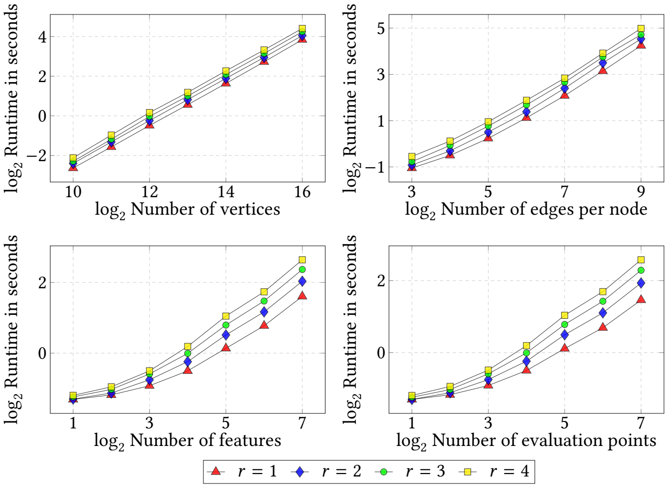

Figure 5: Average runtime of *FEATHER* as a function of node and edge count, number of features and characteristic function evaluation points. The mean runtimes were calculated from 10 repetitions using synthetic ErdÅŚs RÃĺnyi graphs.

Our results support the theoretical runtime complexities discussed in Subsection 3.1. Practically it means that doubling the number of nodes, edges, features or characteristic function evaluation points doubles the expected runtime of the algorithm. Increasing the scale of random walks (considering more hops) increases the runtime. However, for small values of r the quadratic increase in runtime is not evident.

5 CONCLUSIONS AND FUTURE DIRECTIONS

We presented a general notion of characteristic functions defined on attributed graphs. We discussed a specific instance of these – the r-scale random walk weighted characteristic function. We proposed *FEATHER* an efficient algorithm to calculate this characteristic function efficiently on large attributed graphs in linear time to create Euclidean vector space representations of nodes. We proved that *FEATHER* is robust to data corruption and that isomorphic graphs have the same vector space representations. We have shown that *FEATHER* can be interpreted as the forward pass of a neural network and can be used as a differentiable building block for parametric classifiers.

We demonstrated on various real world node and graph classification datasets that *FEATHER* variants are competitive with comparable embedding and graph neural network models. Our transfer learning results support that *FEATHER* models are able to efficiently and robustly transfer knowledge from one graph to another one. The sensitivity analysis of characteristic function based models and node classification results highlight that supervised *FEATHER* models have an edge compared to unsupervised representation creation with characteristic functions. Furthermore, runtime experiments presented show that our proposed algorithm scales linearly with the input size such as number of nodes, edges and features in practice.

As a future direction we would like to point out that the forward pass of the *FEATHER* algorithm could be incorporated in temporal, multiplex and heterogeneous graph neural network models to serve as a multi-scale vertex feature extraction block. Moreover, one could define a characteristic function based node feature pooling where the node feature aggregation is a learned permutation invariant function. Finally, our evaluation of the proposed algorithms was limited to social networks and web graphs – testing it on biological networks and other types of datasets could be an important further direction.

ACKNOWLEDGEMENTS

Benedek Rozemberczki was supported by the Centre for Doctoral Training in Data Science, funded by EPSRC (grant EP/L016427/1). We would also like to thank Rayna Andreeva, Maria Astefanoaei, Tyler Derr, Abhirup Ghosh, Anupam Mediratta and Lauren Watson for their useful comments.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *International Conference on Machine Learning*.

[2] Nesreen K Ahmed, Ryan A Rossi, John Boaz Lee, Theodore L Willke, Rong Zhou, Xiangnan Kong, and Hoda Eldardiry. 2019. role2vec: Role-based network embeddings. In *Proc. DLG KDD*.

[3] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. ACM, 891–900.

[4] Hong Chen and Hisashi Koga. 2019. GL2vec: Graph Embedding Enriched by Line Graphs with Edge Features. In *International Conference on Neural Information Processing*. Springer, 3–14.

[5] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *International Conference on Knowledge Discovery and Data Mining*.

[6] Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F Werneck. 2014. Distance-Based Influence in Networks: Computation and Maximization. *arXiv preprint arXiv:1410.6976* (2014).

[7] Anirban DasGupta. 2011. *Characteristic Functions and Applications*. 293–322.

[8] Nathan de Lara and Pineau Edouard. 2018. A simple baseline algorithm for graph classification. In *Advances in Neural Information Processing Systems*.

[9] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. 2014. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*. 1646–1654.

[10] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. 2018. Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1320–1329.

[11] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[12] Feng Gao, Guy Wolf, and Matthew Hirn. 2019. Geometric Scattering for Graph Data Analysis. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97. 2122–2131.

[13] Hongyang Gao and Shuiwang Ji. 2019. Graph U-nets. In *Proceedings of The 36th International Conference on Machine Learning*.

[14] Thomas Gärtner, Peter Flach, and Stefan Wrobel. 2003. On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*. Springer, 129–143.

[15] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[16] N. Halko, P. G. Martinsson, and J. A. Tropp. 2011. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Rev.* 53, 2 (2011), 217âĂŞ288.

[17] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*.

[18] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. 2012. Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1231–1239.

[19] Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. 2004. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 158–167.

[20] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.

[21] Diederik Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

[22] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

[23] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *International Conference on Learning Representations*.

[24] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-Attention Graph Pooling. In *Proceedings of the 36th International Conference on Machine Learning*.

[25] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. 2018. Attributed Social Network Embedding. *IEEE Transactions on Knowledge and Data Engineering* 30, 12 (2018), 2257–2270.

[26] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, and Yang Liu. 2017. graph2vec: Learning distributed representations of graphs. (2017).

[27] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 1105–1114.

[28] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[29] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[30] Bryan Perozzi, Vivek Kulkarni, Haochen Chen, and Steven Skiena. 2017. Don't Walk, Skip!: online learning of multi-scale network embeddings. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*. ACM, 258–265.

[31] Bryan Perozzi and Steven Skiena. 2015. Exact age prediction in social networks. In *Proceedings of the 24th International Conference on World Wide Web*. 91–92.

[32] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 459–467.

[33] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2019. Multi-scale Attributed Node Embedding. *arXiv preprint arXiv:1909.13021* (2019).

[34] Benedek Rozemberczki, Ryan Davies, Rik Sarkar, and Charles Sutton. 2019. GEMSEC: Graph Embedding with Self Clustering. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2019*. ACM, 65–72.

[35] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. 2020. Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. In *Proceedings of the 29th ACM International on Conference on Information and Knowledge Management (CIKM '20)*. ACM.

[36] Benedek Rozemberczki and Rik Sarkar. 2018. Fast Sequence-Based Embedding with Diffusion Graphs. In *International Workshop on Complex Networks*. Springer, 99–107.

[37] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12, 77 (2011), 2539–2561.

[38] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15, 1 (2014), 1929âĂŞ1958.

[39] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. 1067–1077.

[40] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. 2018. Netlsd: hearing the shape of a graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2347–2356.

[41] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[42] Saurabh Verma and Zhi-Li Zhang. 2017. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems*. 88–98.

[43] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *International Conference on Machine Learning*.

[44] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. 2015. Network representation learning with rich text information. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[45] Hong Yang, Shirui Pan, Peng Zhang, Ling Chen, Defu Lian, and Chengqi Zhang. 2018. Binarized attributed network embedding. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1476–1481.

[46] Shuang Yang and Bo Yang. 2018. Enhanced Network Embedding with Text Information. In *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, 326–331.

[47] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2018. SINE: Scalable Incomplete Network Embedding. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 737–746.

[48] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An End-to-End Deep Learning Architecture for Graph Classification. In *AAAI*. 4438–4445.