DeepWalk: Online Learning of Social Representations

Bryan Perozzi
Stony Brook University
Department of Computer Science

Rami Al-Rfou
Stony Brook University
Department of Computer Science

Steven Skiena
Stony Brook University
Department of Computer Science

{bperozzi, ralrfou, skiena}@cs.stonybrook.edu

ABSTRACT

We present DeepWalk, a novel approach for learning latent representations of vertices in a network. These latent representations encode social relations in a continuous vector space, which is easily exploited by statistical models. DeepWalk generalizes recent advancements in language modeling and unsupervised feature learning (or *deep learning*) from sequences of words to graphs.

DeepWalk uses local information obtained from truncated random walks to *learn* latent representations by treating walks as the equivalent of sentences. We demonstrate DeepWalk's latent representations on several multi-label network classification tasks for social networks such as BlogCatalog, Flickr, and YouTube. Our results show that DeepWalk outperforms challenging baselines which are allowed a global view of the network, especially in the presence of missing information. DeepWalk's representations can provide F_1 scores up to 10% higher than competing methods when labeled data is sparse. In some experiments, DeepWalk's representations are able to outperform all baseline methods while using 60% less training data.

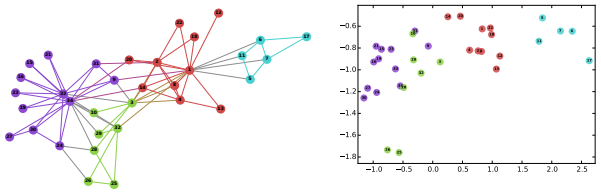
DeepWalk is also scalable. It is an online learning algorithm which builds useful incremental results, and is trivially parallelizable. These qualities make it suitable for a broad class of real world applications such as network classification, and anomaly detection.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications - Data Mining; I.2.6 [**Artificial Intelligence**]: Learning; I.5.1 [**Pattern Recognition**]: Model - Statistical

1. INTRODUCTION

The sparsity of a network representation is both a strength and a weakness. Sparsity enables the design of efficient discrete algorithms, but can make it harder to generalize in statistical learning. Machine learning applications in networks (such as network classification [15, 37], content recommendation [11], anomaly detection [5], and missing link prediction [22]) must be able to deal with this sparsity in order to survive.



(a) Input: Karate Graph (b) Output: Representation

Figure 1: Our proposed method *learns* a latent space representation of social interactions in $\mathbb{R}^d$. The learned representation encodes community structure so it can be easily exploited by standard classification methods. Here, our method is used on Zachary's Karate network [44] to generate a latent representation in $\mathbb{R}^2$. Note the correspondence between community structure in the input graph and the embedding. Vertex colors represent a modularity-based clustering of the input graph.

In this paper we introduce *deep learning* (unsupervised feature learning) [2] techniques, which have proven successful in natural language processing, into network analysis for the first time. We develop an algorithm (DeepWalk) that learns *social representations* of a graph's vertices, by modeling a stream of short random walks. Social representations are latent features of the vertices that capture neighborhood similarity and community membership. These latent representations encode social relations in a continuous vector space with a relatively small number of dimensions. DeepWalk generalizes neural language models to process a special language composed of a set of randomly-generated walks. These neural language models have been used to capture the semantic and syntactic structure of human language [6], and even logical analogies [28].

DeepWalk takes a graph as input and produces a latent representation as an output. The result of applying our method to the well-studied Karate network is shown in Figure 1. The graph, as typically presented by force-directed layouts, is shown in Figure 1a. Figure 1b shows the output of our method with 2 latent dimensions. Beyond the striking similarity, we note that linearly separable portions of (1b) correspond to clusters found through modularity maximization in the input graph (1a) (shown as vertex colors).

To demonstrate DeepWalk's potential in real world sce-

narios, we evaluate its performance on challenging multi-label network classification problems in large heterogeneous graphs. In the relational classification problem, the links between feature vectors violate the traditional *i.i.d.* assumption. Techniques to address this problem typically use approximate inference techniques [31, 35] to leverage the dependency information to improve classification results. We distance ourselves from these approaches by learning label-independent representations of the graph. Our representation quality is not influenced by the choice of labeled vertices, so they can be shared among tasks.

DeepWalk outperforms other latent representation methods for creating *social dimensions* [39, 41], especially when labeled nodes are scarce. Strong performance with our representations is possible with very simple linear classifiers (e.g. logistic regression). Our representations are general, and can be combined with any classification method (including iterative inference methods). DeepWalk achieves all of that while being an online algorithm that is trivially parallelizable.

Our contributions are as follows:

- We introduce deep learning as a tool to analyze graphs, to build robust representations that are suitable for statistical modeling. DeepWalk learns structural regularities present within short random walks.
- We extensively evaluate our representations on multi-label classification tasks on several social networks. We show significantly increased classification performance in the presence of label sparsity, getting improvements 5%-10% of Micro F_1, on the sparsest problems we consider. In some cases, DeepWalk's representations can outperform its competitors even when given 60% less training data.
- We demonstrate the scalability of our algorithm by building representations of web-scale graphs, (such as YouTube) using a parallel implementation. Moreover, we describe the minimal changes necessary to build a streaming version of our approach.

The rest of the paper is arranged as follows. In Sections 2 and 3, we discuss the problem formulation of classification in data networks, and how it relates to our work. In Section 4 we present DeepWalk, our approach for Social Representation Learning. We outline ours experiments in Section 5, and present their results in Section 6. We close with a discussion of related work in Section 7, and our conclusions.

2. PROBLEM DEFINITION

We consider the problem of classifying members of a social network into one or more categories. More formally, let $G = (V, E)$, where V are the members of the network, and E be its edges, $E \subseteq (V \times V)$. Given a partially labeled social network $G_L = (V, E, X, Y)$, with attributes $X \in \mathbb{R}^{|V| \times S}$ where S is the size of the feature space for each attribute vector, and $Y \in \mathbb{R}^{|V| \times |\mathcal{Y}|}$, $\mathcal{Y}$ is the set of labels.

In a traditional machine learning classification setting, we aim to learn a hypothesis H that maps elements of X to the labels set $\mathcal{Y}$. In our case, we can utilize the significant information about the dependence of the examples embedded in the structure of G to achieve superior performance.

In the literature, this is known as the relational classification (or the *collective classification* problem [37]). Traditional approaches to relational classification pose the problem as an inference in an undirected Markov network, and then use iterative approximate inference algorithms (such as the iterative classification algorithm [31], Gibbs Sampling [14], or label relaxation [18]) to compute the posterior distribution of labels given the network structure.

We propose a different approach to capture the network topology information. Instead of mixing the label space as part of the feature space, we propose an unsupervised method which learns features that capture the graph structure *independent* of the labels' distribution.

This separation between the structural representation and the labeling task avoids cascading errors, which can occur in iterative methods [33]. Moreover, the same representation can be used for multiple classification problems concerning that network.

Our goal is to learn $X_E \in \mathbb{R}^{|V| \times d}$, where d is small number of latent dimensions. These low-dimensional representations are distributed; meaning each social phenomena is expressed by a subset of the dimensions and each dimension contributes to a subset of the social concepts expressed by the space.

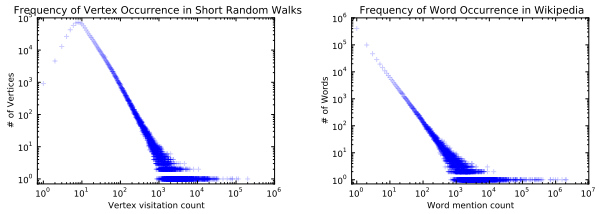
Using these structural features, we will augment the attributes space to help the classification decision. These features are general, and can be used with any classification algorithm (including iterative methods). However, we believe that the greatest utility of these features is their easy integration with simple machine learning algorithms. They scale appropriately in real-world networks, as we will show in Section 6.

3. LEARNING SOCIAL REPRESENTATIONS

We seek learning social representations with the following characteristics:

- **Adaptability** - Real social networks are constantly evolving; new social relations should not require repeating the learning process all over again.
- **Community aware** - The distance between latent dimensions should represent a metric for evaluating social similarity between the corresponding members of the network. This allows generalization in networks with homophily.
- **Low dimensional** - When labeled data is scarce, low-dimensional models generalize better, and speed up convergence and inference.
- **Continuous** - We require latent representations to model partial community membership in continuous space. In addition to providing a nuanced view of community membership, a continuous representation has smooth decision boundaries between communities which allows more robust classification.

Our method for satisfying these requirements learns representation for vertices from a stream of short random walks, using optimization techniques originally designed for language modeling. Here, we review the basics of both random walks and language modeling, and describe how their combination satisfies our requirements.



(a) YouTube Social Graph (b) Wikipedia Article Text

Figure 2: The power-law distribution of vertices appearing in short random walks (2a) follows a power-law, much like the distribution of words in natural language (2b).

3.1 Random Walks

We denote a random walk rooted at vertex v_i as $\mathcal{W}_{v_i}$. It is a stochastic process with random variables $\mathcal{W}_{v_i}^1, \mathcal{W}_{v_i}^2, \ldots, \mathcal{W}_{v_i}^k$ such that $\mathcal{W}_{v_i}^{k+1}$ is a vertex chosen at random from the neighbors of vertex v_k. Random walks have been used as a similarity measure for a variety of problems in content recommendation [11] and community detection [1]. They are also the foundation of a class of *output sensitive* algorithms which use them to compute local community structure information in time sublinear to the size of the input graph [38].

It is this connection to local structure that motivates us to use a *stream* of short random walks as our basic tool for extracting information from a network. In addition to capturing community information, using random walks as the basis for our algorithm gives us two other desirable properties. First, local exploration is easy to parallelize. Several random walkers (in different threads, processes, or machines) can simultaneously explore different parts of the same graph. Secondly, relying on information obtained from short random walks make it possible to accommodate small changes in the graph structure without the need for global recomputation. We can iteratively update the learned model with new random walks from the changed region in time sub-linear to the entire graph.

3.2 Connection: Power laws

Having chosen online random walks as our primitive for capturing graph structure, we now need a suitable method to capture this information. If the degree distribution of a connected graph follows a power law (is *scale-free*), we observe that the frequency which vertices appear in the short random walks will also follow a power-law distribution.

Word frequency in natural language follows a similar distribution, and techniques from language modeling account for this distributional behavior. To emphasize this similarity we show two different power-law distributions in Figure 2. The first comes from a series of short random walks on a scale-free graph, and the second comes from the text of 100,000 articles from the English Wikipedia.

A core contribution of our work is the idea that techniques which have been used to model natural language (where the symbol frequency follows a power law distribution (or *Zipf's law*)) can be re-purposed to model community structure in networks. We spend the rest of this section reviewing the growing work in language modeling, and transforming it to learn representations of vertices which satisfy our criteria.

3.3 Language Modeling

The goal of language modeling is estimate the likelihood of a specific sequence of words appearing in a corpus. More formally, given a sequence of words

$$W_1^n = (w_0, w_1, \cdots, w_n)$$

where $w_i \in \mathcal{V}$ ($\mathcal{V}$ is the vocabulary), we would like to maximize the $\Pr(w_n | w_0, w_1, \cdots, w_{n-1})$ over all the training corpus.

Recent work in representation learning has focused on using probabilistic neural networks to build general representations of words which extend the scope of language modeling beyond its original goals.

In this work, we present a generalization of language modeling to explore the graph through a stream of short random walks. These walks can be thought of short sentences and phrases in a special language. The direct analog is to estimate the likelihood of observing vertex v_i given all the previous vertices visited so far in the random walk.

$$\Pr\left(v_i \mid (v_1, v_2, \cdots, v_{i-1})\right)$$

Our goal is to learn a latent representation, not only a probability distribution of node co-occurrences, and so we introduce a mapping function $\Phi\colon v \in V \mapsto \mathbb{R}^{|V| \times d}$. This mapping Φ represents the latent social representation associated with each vertex v in the graph. (In practice, we represent Φ by a $|V| \times d$ matrix of free parameters, which will serve later on as our X_E.) The problem then, is to estimate the likelihood:

$$\Pr\left(v_i \mid \left(\Phi(v_1), \Phi(v_2), \cdots, \Phi(v_{i-1})\right)\right) \tag{1}$$

However as the walk length grows, computing this objective function becomes unfeasible.

A recent relaxation in language modeling [26, 27] turns the prediction problem on its head. First, instead of using the context to predict a missing word, it uses one word to predict the context. Secondly, the context is composed of the words appearing to right side of the given word as well as the left side. Finally, it removes the ordering constraint on the problem. Instead, the model is required to maximize the probability of any word appearing in the context without the knowledge of its offset from the given word.

In terms of vertex representation modeling, this yields the optimization problem:

$$\underset{\Phi}{\text{minimize}} \quad -\log \Pr\left(\{v_{i-w}, \cdots, v_{i-1}, v_{i+1}, \cdots, v_{i+w}\} \mid \Phi(v_i)\right) \tag{2}$$

We find these relaxations are particularly desirable for social representation learning. First, the order independence assumption better captures a sense of 'nearness' that is provided by random walks. Moreover, this relaxation is quite useful for speeding up the training time by building small models as one vertex is given at a time.

Solving the optimization problem from Eq. 2 builds representations that capture the shared similarities in local graph structure between vertices. Vertices which have similar neighborhoods will acquire similar representations (encoding cocitation similarity), and allowing generalization on machine learning tasks.

By combining both truncated random walks and neural language models we formulate a method which satisfies all

Algorithm 1 DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

1: Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
2: Build a binary Tree T from V
3: **for** $i = 0$ to γ **do**
4: $\quad \mathcal{O} = \text{Shuffle}(V)$
5: $\quad$ **for each** $v_i \in \mathcal{O}$ **do**
6: $\quad\quad \mathcal{W}_{v_i} = RandomWalk(G, v_i, t)$
7: $\quad\quad$ SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
8: $\quad$ **end for**
9: **end for**

of our desired properties. This method generates representations of social networks that are low-dimensional, and exist in a continuous vector space. Its representations encode latent forms of community membership, and because the method outputs useful intermediate representations, it can adapt to changing network topology.

4. METHOD

In this section we discuss the main components of our algorithm. We also present several variants of our approach and discuss their merits.

4.1 Overview

As in any language modeling algorithm, the only required input is a corpus and a vocabulary $\mathcal{V}$. DeepWalk considers a set of short truncated random walks its own corpus, and the graph vertices as its own vocabulary ($\mathcal{V} = V$). While it is beneficial to know the V and the frequency distribution of vertices in the random walks ahead of the training, it is not necessary for the algorithm to work as we will show in 4.2.2.

4.2 Algorithm: DeepWalk

The algorithm consists of two main components; first a random walk generator and second an update procedure.

The random walk generator takes a graph G and samples uniformly a random vertex v_i as the root of the random walk $\mathcal{W}_{v_i}$. A walk samples uniformly from the neighbors of the last vertex visited until the maximum length (t) is reached. While we set the length of our random walks in the experiments to be fixed, there is no restriction for the random walks to be of the same length. These walks could have restarts (i.e. a teleport probability of returning back to their root), but our preliminary results did not show any advantage of using restarts. In practice, our implementation specifies a number of random walks γ of length t to start at each vertex.

Lines 3-9 in Algorithm 1 shows the core of our approach. The outer loop specifies the number of times, γ, which we should start random walks at each vertex. We think of each iteration as making a 'pass' over the data and sample one walk per node during this pass. At the start of each pass we generate a random ordering to traverse the vertices. This is not strictly required, but is well-known to speed up the convergence of stochastic gradient descent.

Algorithm 2 SkipGram(Φ, $\mathcal{W}_{v_i}$, w)

1: **for each** $v_j \in \mathcal{W}_{v_i}$ **do**
2: $\quad$ **for each** $u_k \in \mathcal{W}_{v_i}[j - w : j + w]$ **do**
3: $\quad\quad J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
4: $\quad\quad \Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
5: $\quad$ **end for**
6: **end for**

In the inner loop, we iterate over all the vertices of the graph. For each vertex v_i we generate a random walk $|\mathcal{W}_{v_i}| = t$, and then use it to update our representations (Line 7). We use the SkipGram algorithm [26] to update these representations in accordance with our objective function in Eq. 2.

4.2.1 SkipGram

SkipGram is a language model that maximizes the co-occurrence probability among the words that appear within a window, w, in a sentence [26].

Algorithm 2 iterates over all possible collocations in random walk that appear within the window w (lines 1-2). For each, we map each vertex v_j to its current representation vector $\Phi(v_j) \in \mathbb{R}^d$ (See Figure 3b). Given the representation of v_j, we would like to maximize the probability of its neighbors in the walk (line 3). We can learn such posterior distribution using several choices of classifiers. For example, modeling the previous problem using logistic regression would result in a huge number of labels that is equal to $|V|$ which could be in millions or billions. Such models require large amount of computational resources that could span a whole cluster of computers [3]. To speed the training time, Hierarchical Softmax [29,30] can be used to approximate the probability distribution.

4.2.2 Hierarchical Softmax

Given that $u_k \in V$, calculating $\Pr(u_k \mid \Phi(v_j))$ in line 3 is not feasible. Computing the partition function (normalization factor) is expensive. If we assign the vertices to the leaves of a binary tree, the prediction problem turns into maximizing the probability of a specific path in the tree (See Figure 3c). If the path to vertex u_k is identified by a sequence of tree nodes $(b_0, b_1, \ldots, b_{\lceil \log |V| \rceil})$, ($b_0$ = root, $b_{\lceil \log |V| \rceil} = u_k$) then

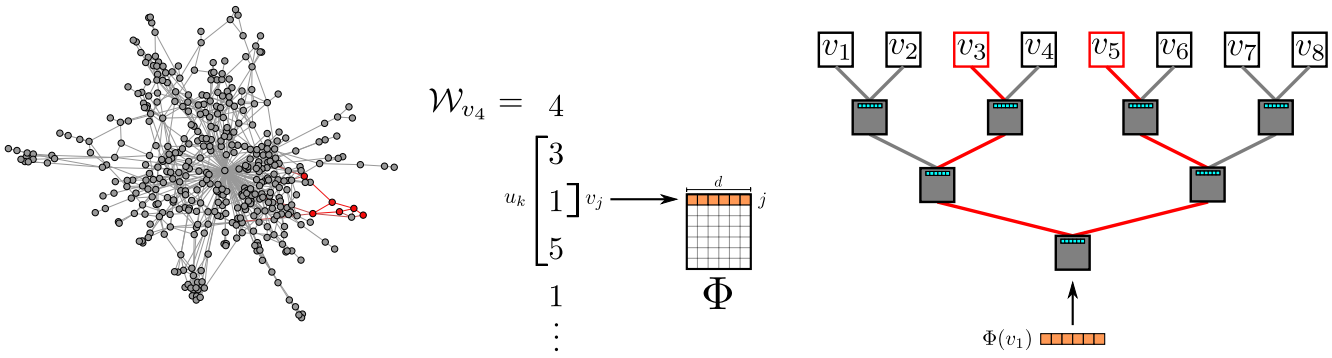
$$\Pr(u_k \mid \Phi(v_j)) = \prod_{l=1}^{\lceil \log |V| \rceil} \Pr(b_l \mid \Phi(v_j))$$

Now, $\Pr(b_l \mid \Phi(v_j))$ could be modeled by a binary classifier that is assigned to the parent of the node b_l. This reduces the computational complexity of calculating $\Pr(u_k \mid \Phi(v_j))$ from $O(|V|)$ to $O(\log |V|)$.

We can speed up the training process further, by assigning shorter paths to the frequent vertices in the random walks. Huffman coding is used to reduce the access time of frequent elements in the tree.

4.2.3 Optimization

The model parameter set is $\{\Phi, T\}$ where the size of each is $O(d|V|)$. Stochastic gradient descent (SGD) [4] is used to optimize these parameters (Line 4, Algorithm 2). The derivatives are estimated using the back-propagation algorithm. The learning rate α for SGD is initially set to 2.5% at the beginning of the training and then decreased linearly



(a) Random walk generation. (b) Representation mapping. (c) Hierarchical Softmax.

Figure 3: Overview of DeepWalk. We slide a window of length $2w+1$ over the random walk $\mathcal{W}_{v_4}$, mapping the central vertex v_1 to its representation $\Phi(v_1)$. Hierarchical Softmax factors out $\Pr(v_3 \mid \Phi(v_1))$ and $\Pr(v_5 \mid \Phi(v_1))$ over sequences of probability distributions corresponding to the paths starting at the root and ending at v_3 and v_5. The representation Φ is updated to maximize the probability of v_1 co-occurring with its context $\{v_3, v_5\}$.

with the number of vertices that are seen so far.

4.3 Parallelizability

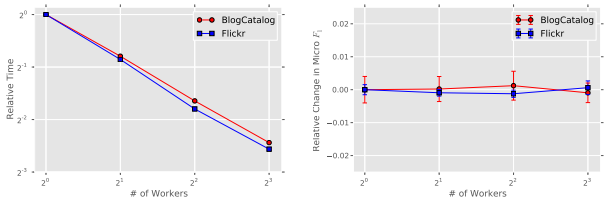
As shown in Figure 2 the frequency distribution of vertices in random walks of social network and words in a language both follow a power law. This results in a long tail of infrequent vertices, therefore, the updates that affect Φ will be sparse in nature. This allows us to use asynchronous version of stochastic gradient descent (ASGD), in the multi-worker case. Given that our updates are sparse and we do not acquire a lock to access the model shared parameters, ASGD will achieve an optimal rate of convergence [36]. While we run experiments on one machine using multiple threads, it has been demonstrated that this technique is highly scalable, and can be used in very large scale machine learning [8]. Figure 4 presents the effects of parallelizing DeepWalk. It shows the speed up in processing BlogCatalog and Flickr networks is consistent as we increase the number of workers to 8 (Figure 4a). It also shows that there is no loss of predictive performance relative to the running DeepWalk serially (Figure 4b).

4.4 Algorithm Variants

Here we discuss some variants of our proposed method, which we believe may be of interest.

4.4.1 Streaming

One interesting variant of this method is a *streaming* approach, which could be implemented without knowledge of the entire graph. In this variant small walks from the graph are passed directly to the representation learning code, and the model is updated directly. Some modifications to the learning process will also be necessary. First, using a decaying learning rate will no longer be possible. Instead, we can initialize the learning rate α to a small constant value. This will take longer to learn, but may be worth it in some applications. Second, we cannot necessarily build a tree of parameters any more. If the cardinality of V is known (or can be bounded), we can build the Hierarchical Softmax tree for that maximum value. Vertices can be assigned to one of the remaining leaves when they are first seen. If we have the ability to estimate the vertex frequency a priori, we can also still use Huffman coding to decrease frequent element access times.



(a) Running Time (b) Performance

Figure 4: Effects of parallelizing DeepWalk

4.4.2 Non-random walks

Some graphs are created as a by-product of agents interacting with a sequence of elements (e.g. users' navigation of pages on a website). When a graph is created by such a stream of *non-random* walks, we can use this process to feed the modeling phase directly. Graphs sampled in this way will not only capture information related to network structure, but also to the frequency at which paths are traversed.

In our view, this variant also encompasses language modeling. Sentences can be viewed as purposed walks through an appropriately designed language network, and language models like SkipGram are designed to capture this behavior.

This approach can be combined with the streaming variant (Section 4.4.1) to train features on a continually evolving network without ever explicitly constructing the entire graph. Maintaining representations with this technique could enable web-scale classification without the hassles of dealing with a web-scale graph.

5. EXPERIMENTAL DESIGN

In this section we provide an overview of the datasets and methods which we will use in our experiments. Code and data to reproduce our results will be available at the first author's website.

5.1 Datasets

Name	BlogCatalog	Flickr	YouTube
$\|V\|$	10,312	80,513	1,138,499
$\|E\|$	333,983	5,899,882	2,990,443
$\|\mathcal{Y}\|$	39	195	47
Labels	Interests	Groups	Groups

Table 1: Graphs used in our experiments.

An overview of the graphs we consider in our experiments is given in Figure 1.

- BlogCatalog [39] is a network of social relationships provided by blogger authors. The labels represent the topic categories provided by the authors.
- Flickr [39] is a network of the contacts between users of the photo sharing website. The labels represent the interest groups of the users such as '*black and white photos*'.
- YouTube [40] is a social network between users of the popular video sharing website. The labels here represent groups of viewers that enjoy common video genres (e.g. *anime* and *wrestling*).

5.2 Baseline Methods

To validate the performance of our approach we compare it against a number of baselines:

- SpectralClustering [41]: This method generates a representation in $\mathbb{R}^d$ from the d-smallest eigenvectors of $\widetilde{\mathcal{L}}$, the normalized graph Laplacian of G. Utilizing the eigenvectors of $\widetilde{\mathcal{L}}$ implicitly assumes that graph cuts will be useful for classification.
- Modularity [39]: This method generates a representation in $\mathbb{R}^d$ from the top-d eigenvectors of B, the Modularity matrix of G. The eigenvectors of B encode information about modular graph partitions of G [34]. Using them as features assumes that modular graph partitions will be useful for classification.
- EdgeCluster [40]: This method uses k-means clustering to cluster the adjacency matrix of G. Its has been shown to perform comparably to the Modularity method, with the added advantage of scaling to graphs which are too large for spectral decomposition.
- wvRN [24]: The weighted-vote Relational Neighbor is a relational classifier. Given the neighborhood $\mathcal{N}_i$ of vertex v_i, wvRN estimates $\Pr(y_i|\mathcal{N}_i)$ with the (appropriately normalized) weighted mean of its neighbors (i.e $\Pr(y_i|\mathcal{N}_i) = \frac{1}{Z}\sum_{v_j \in \mathcal{N}_i} w_{ij} \Pr(y_j \mid \mathcal{N}_j)$). It has shown surprisingly good performance in real networks, and has been advocated as a sensible relational classification baseline [25].
- Majority: This naïve method simply chooses the most frequent labels in the training set.

6. EXPERIMENTS

In this section we present an experimental analysis of our method. We thoroughly evaluate it on a number of multi-label classification tasks, and analyze its sensitivity across several parameters.

6.1 Multi-Label Classification

To facilitate the comparison between our method and the relevant baselines, we use the exact same datasets and experimental procedure as in [39, 40]. Specifically, we randomly sample a portion (T_R) of the labeled nodes, and use them as training data. The rest of the nodes are used as test. We repeat this process 10 times, and report the average performance in terms of both Macro-F_1 and Micro-F_1. When possible we report the original results [39, 40] here directly.

For all models we use a one-vs-rest logistic regression implemented by LibLinear [10] for classification. We present results for DeepWalk with ($\gamma = 80$, $w = 10$, $d = 128$). The results for (SpectralClustering, Modularity, EdgeCluster) use Tang and Liu's preferred dimensionality, $d = 500$.

6.1.1 BlogCatalog

In this experiment we increase the training ratio (T_R) on the BlogCatalog network from 10% to 90%. Our results are presented in Table 2. Numbers in bold represent the highest performance in each column.

DeepWalk performs consistently better than EdgeCluster, Modularity, and wvRN. In fact, when trained with only 20% of the nodes labeled, DeepWalk performs better than these approaches when they are given 90% of the data. The performance of SpectralClustering proves much more competitive, but DeepWalk still outperforms when labeled data is sparse on both Macro-F_1 ($T_R \leq 20\%$) and Micro-F_1 ($T_R \leq 60\%$).

This strong performance when only small fractions of the graph are labeled is a core strength of our approach. In the following experiments, we investigate the performance of our representations on even more sparsely labeled graphs.

6.1.2 Flickr

In this experiment we vary the training ratio (T_R) on the Flickr network from 1% to 10%. This corresponds to having approximately 800 to 8,000 nodes labeled for classification in the entire network. Table 3 presents our results, which are consistent with the previous experiment. DeepWalk outperforms all baselines by at least 3% with respect to Micro-F_1. Additionally, its Micro-F_1 performance when only 3% of the graph is labeled beats all other methods even when they have been given 10% of the data. In other words, DeepWalk can outperform the baselines with 60% less training data. It also performs quite well in Macro-F_1, initially performing close to SpectralClustering, but distancing itself to a 1% improvement.

6.1.3 YouTube

The YouTube network is considerably larger than the previous ones we have experimented on, and its size prevents two of our baseline methods (SpectralClustering and Modularity) from running on it. It is much closer to a real world graph than those we have previously considered.

The results of varying the training ratio (T_R) from 1% to 10% are presented in Table 4. They show that DeepWalk significantly outperforms the scalable baseline for creating graph representations, EdgeCluster. When 1% of the labeled nodes are used for test, the Micro-F_1 improves by 14%. The Macro-F_1 shows a corresponding 10% increase. This lead narrows as the training data increases, but DeepWalk ends with a 3% lead in Micro-F_1, and an impressive 5% improvement in Macro-F_1.

This experiment showcases the performance benefits that

	% Labeled Nodes	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1(%)	DeepWalk	**36.00**	**38.20**	**39.60**	**40.30**	**41.00**	**41.30**	41.50	41.50	42.00
	SpectralClustering	31.06	34.95	37.27	38.93	39.97	40.99	**41.66**	**42.42**	**42.62**
	EdgeCluster	27.94	30.76	31.85	32.99	34.12	35.00	34.63	35.99	36.29
	Modularity	27.35	30.74	31.77	32.97	34.09	36.13	36.08	37.23	38.18
	wvRN	19.51	24.34	25.62	28.82	30.37	31.81	32.19	33.33	34.28
	Majority	16.51	16.66	16.61	16.70	16.91	16.99	16.92	16.49	17.26
Macro-F1(%)	DeepWalk	**21.30**	**23.80**	25.30	26.30	27.30	27.60	27.90	28.20	28.90
	SpectralClustering	19.14	23.57	**25.97**	**27.46**	**28.31**	**29.46**	**30.13**	**31.38**	**31.78**
	EdgeCluster	16.16	19.16	20.48	22.00	23.00	23.64	23.82	24.61	24.92
	Modularity	17.36	20.00	20.80	21.85	22.65	23.41	23.89	24.20	24.97
	wvRN	6.25	10.13	11.64	14.24	15.86	17.18	17.98	18.86	19.57
	Majority	2.52	2.55	2.52	2.58	2.58	2.63	2.61	2.48	2.62

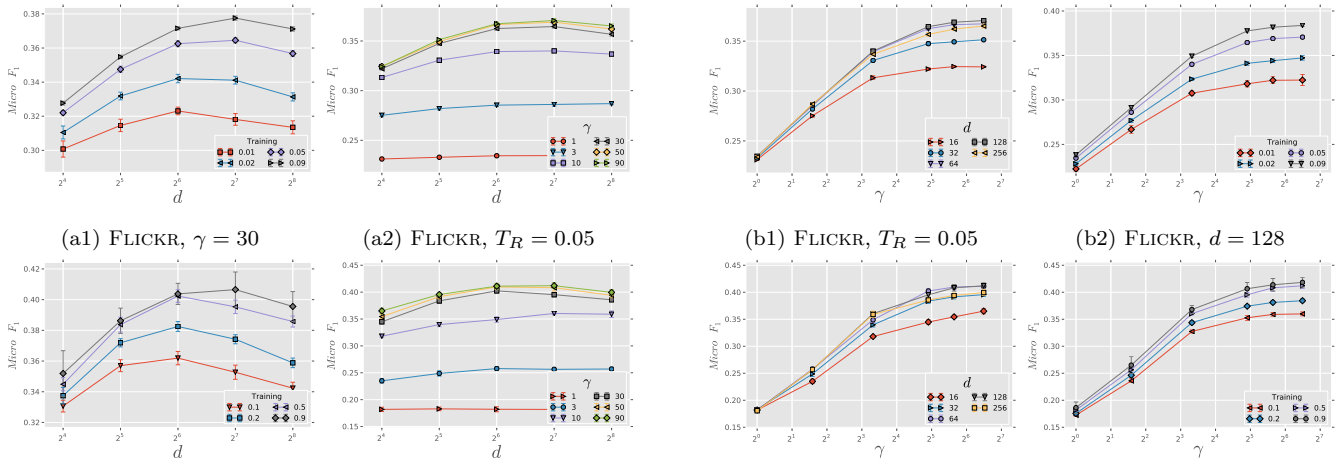
Table 2: Multi-label classification results in BlogCatalog

	% Labeled Nodes	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F1(%)	DeepWalk	**32.4**	**34.6**	**35.9**	**36.7**	**37.2**	**37.7**	**38.1**	**38.3**	**38.5**	**38.7**
	SpectralClustering	27.43	30.11	31.63	32.69	33.31	33.95	34.46	34.81	35.14	35.41
	EdgeCluster	25.75	28.53	29.14	30.31	30.85	31.53	31.75	31.76	32.19	32.84
	Modularity	22.75	25.29	27.3	27.6	28.05	29.33	29.43	28.89	29.17	29.2
	wvRN	17.7	14.43	15.72	20.97	19.83	19.42	19.22	21.25	22.51	22.73
	Majority	16.34	16.31	16.34	16.46	16.65	16.44	16.38	16.62	16.67	16.71
Macro-F1(%)	DeepWalk	**14.0**	17.3	**19.6**	**21.1**	**22.1**	**22.9**	**23.6**	**24.1**	**24.6**	**25.0**
	SpectralClustering	13.84	**17.49**	19.44	20.75	21.60	22.36	23.01	23.36	23.82	24.05
	EdgeCluster	10.52	14.10	15.91	16.72	18.01	18.54	19.54	20.18	20.78	20.85
	Modularity	10.21	13.37	15.24	15.11	16.14	16.64	17.02	17.1	17.14	17.12
	wvRN	1.53	2.46	2.91	3.47	4.95	5.56	5.82	6.59	8.00	7.26
	Majority	0.45	0.44	0.45	0.46	0.47	0.44	0.45	0.47	0.47	0.47

Table 3: Multi-label classification results in Flickr

	% Labeled Nodes	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F1(%)	DeepWalk	**37.95**	**39.28**	**40.08**	**40.78**	**41.32**	**41.72**	**42.12**	**42.48**	**42.78**	**43.05**
	SpectralClustering	—	—	—	—	—	—	—	—	—	—
	EdgeCluster	23.90	31.68	35.53	36.76	37.81	38.63	38.94	39.46	39.92	40.07
	Modularity	—	—	—	—	—	—	—	—	—	—
	wvRN	26.79	29.18	33.1	32.88	35.76	37.38	38.21	37.75	38.68	39.42
	Majority	24.90	24.84	25.25	25.23	25.22	25.33	25.31	25.34	25.38	25.38
Macro-F1(%)	DeepWalk	**29.22**	**31.83**	**33.06**	**33.90**	**34.35**	**34.66**	**34.96**	**35.22**	**35.42**	**35.67**
	SpectralClustering	—	—	—	—	—	—	—	—	—	—
	EdgeCluster	19.48	25.01	28.15	29.17	29.82	30.65	30.75	31.23	31.45	31.54
	Modularity	—	—	—	—	—	—	—	—	—	—
	wvRN	13.15	15.78	19.66	20.9	23.31	25.43	27.08	26.48	28.33	28.89
	Majority	6.12	5.86	6.21	6.1	6.07	6.19	6.17	6.16	6.18	6.19

Table 4: Multi-label classification results in YouTube



(a1) FLICKR, $\gamma = 30$ (a2) FLICKR, $T_R = 0.05$ (b1) FLICKR, $T_R = 0.05$ (b2) FLICKR, $d = 128$

(a3) BLOGCATALOG, $\gamma = 30$ (a4) BLOGCATALOG, $T_R = 0.5$ (b3) BLOGCATALOG, $T_R = 0.5$ (b4) BLOGCATALOG, $d = 128$

(a) Stability over dimensions, d (a) Stability over number of walks, γ

Figure 5: Parameter Sensitivity Study

can occur from using social representation learning for multi-label classification. DEEPWALK, can scale to large graphs, and performs exceedingly well in such a sparsely labeled environment.

6.2 Parameter Sensitivity

In order to evaluate how changes to the parameterization of DEEPWALK effect its performance on classification tasks, we conducted experiments on two multi-label classifications tasks (FLICKR, and BLOGCATALOG). For this test, we have fixed the window size and the walk length to sensible values ($w = 10, t = 40$) which should emphasize local structure. We then vary the number of latent dimensions (d), the number of walks started per vertex (γ), and the amount of training data available (T_R) to determine their impact on the network classification performance.

6.2.1 *Effect of Dimensionality*

Figure 5a shows the effects of increasing the number of latent dimensions available to our model.

Figures 5a1 and 5a3 examine the effects of varying the dimensionality and training rate. The performance is quite consistent between both FLICKR and BLOGCATALOG and show that the optimal dimensionality for a model is dependent on the number of training examples. (Note that 1% of FLICKR has approximately as many labeled examples as 10% of BLOGCATALOG).

Figures 5a2 and 5a3 examine the effects of varying the dimensionality and number of walks per vertex. The relative performance between dimensions is relatively stable across different values of γ. These charts have two interesting observations. The first is that there is most of the benefit is accomplished by starting $\gamma = 30$ walks per node in both graphs. The second is that the relative difference between different values of γ is quite consistent between the two graphs. FLICKR has an order of magnitude more edges than BLOGCATALOG, and we find this behavior interesting.

These experiments show that our method can make useful models of various sizes. They also show that the performance of the model depends on the number of random walks it has seen, and the appropriate dimensionality of the model depends on the training examples available.

6.2.2 *Effect of sampling frequency*

Figure 5a shows the effects of increasing γ, the number of random walks that we start from each vertex.

The results are very consistent for different dimensions (Fig. 5b1, Fig. 5b3) and the amount of training data (Fig. 5b2, Fig. 5b4). Initially, increasing γ has a big effect in the results, but this effect quickly slows ($\gamma > 10$). These results demonstrate that we are able to learn meaningful latent representations for vertices after only a small number of random walks.

7. RELATED WORK

The main differences between our proposed method and previous work can be summarized as follows:

1. We *learn* our latent social representations, instead of computing statistics related to centrality [12] or partitioning [41].
2. We do not attempt to extend the classification procedure itself (through collective inference [37] or graph kernels [20]).
3. We propose a scalable online method which uses only local information. Most methods require global information and are offline [16, 39–41].
4. We apply unsupervised representation learning to graphs.

In this section we discuss related work in network classification and unsupervised feature learning.

7.1 Relational Learning

Relational classification (or *collective classification*) methods [14,24,31,35] use links between data items as part of the classification process. Exact inference in the collective classification problem is NP-hard, and solutions have focused on the use of approximate inference algorithm which may not be guaranteed to converge [37].

The most relevant relational classification algorithms to our work incorporate community information by learning clusters [32], by adding edges between nearby nodes [13], by using PageRank [23], or by extending relational classification to take additional features into account [43]. Our work takes a substantially different approach. Instead of a new approximation inference algorithm, we propose a procedure which learns representations of network structure which can then be used by existing inference procedure (including iterative ones).

A number of techniques for generating features from graphs have also been proposed [12, 16, 39–41]. In contrast to these methods, we frame the feature creation procedure as a representation learning problem.

Graph Kernels [42] have been proposed as a way to use relational data as part of the classification process, but are quite slow unless approximated [19]. Our approach is complementary; instead of encoding the structure as part of a kernel function, we learn a representation which allows them to be used directly as features for any classification method.

7.2 Unsupervised Feature Learning

Distributed representations have been proposed to model structural relationship between concepts [17]. These representations are trained by the back-propagation and gradient descent. Computational costs and numerical instability led to these techniques to be abandoned for almost a decade. Recently, distributed computing allowed for larger models to be trained [3], and the growth of data for unsupervised learning algorithms to emerge [9]. Distributed representations usually are trained through neural networks, these networks have made advancements in diverse fields such as computer vision [21], speech recognition [7], and natural language processing [6].

8. CONCLUSIONS

We propose DeepWalk, a novel approach for learning latent social representations of vertices. Using local information from truncated random walks as input, our method learns a representation which encodes structural regularities. Experiments on a variety of different graphs illustrate the effectiveness of our approach on challenging multi-label classification tasks.

As an online algorithm, DeepWalk is also scalable. Our results show that we can create meaningful representations for graphs too large to run spectral methods on. On such large graphs, our method significantly outperforms other methods designed to operate for sparsity. We also show that our approach is parallelizable, allowing workers to update different parts of the model concurrently.

In addition to being effective and scalable, our approach is also an appealing generalization of language modeling. This connection is mutually beneficial. Advances in language modeling may continue to generate improved latent representations for networks. In our view, language modeling is actually sampling from an unobservable language graph. We believe that insights obtained from modeling observable graphs may in turn yield improvements to modeling unobservable ones.

Our future work in the area will focus on investigating this duality further, using our results to improve language modeling, and strengthening the theoretical justifications of the method.

9. REFERENCES

[1] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 475–486. IEEE, 2006.

[2] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. 2013.

[3] Y. Bengio, R. Ducharme, and P. Vincent. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[4] L. Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France, 1991. EC2.

[5] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[6] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[7] G. E. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42, 2012.

[8] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng. Large scale distributed deep networks. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1232–1240. 2012.

[9] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, 2010.

[10] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[11] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *Knowledge and Data Engineering, IEEE Transactions on*, 19(3):355–369, 2007.

[12] B. Gallagher and T. Eliassi-Rad. Leveraging label-independent features for classification in sparsely labeled networks: An empirical study. In *Advances in Social Network Mining and Analysis*, pages 1–19. Springer, 2010.

[13] B. Gallagher, H. Tong, T. Eliassi-Rad, and C. Faloutsos. Using ghost edges for classification in sparsely labeled networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 256–264, New York, NY, USA, 2008. ACM.

[14] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):721–741, 1984.

[15] L. Getoor and B. Taskar. *Introduction to statistical relational learning*. MIT press, 2007.

[16] K. Henderson, B. Gallagher, L. Li, L. Akoglu, T. Eliassi-Rad, H. Tong, and C. Faloutsos. It's who you know: Graph mining using recursive structural features. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 663–671, New York, NY, USA, 2011. ACM.

[17] G. E. Hinton. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, pages 1–12. Amherst, MA, 1986.

[18] R. A. Hummel and S. W. Zucker. On the foundations of relaxation labeling processes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (3):267–287, 1983.

[19] U. Kang, H. Tong, and J. Sun. Fast random walk graph kernel. In *SDM*, pages 828–838, 2012.

[20] R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *ICML*, volume 2, pages 315–322, 2002.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, volume 1, page 4, 2012.

[22] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

[23] F. Lin and W. Cohen. Semi-supervised classification of network data using very few labels. In *Advances in Social Networks Analysis and Mining (ASONAM), 2010 International Conference on*, pages 192–199, Aug 2010.

[24] S. A. Macskassy and F. Provost. A simple relational classifier. In *Proceedings of the Second Workshop on Multi-Relational Data Mining (MRDM-2003) at KDD-2003*, pages 64–76, 2003.

[25] S. A. Macskassy and F. Provost. Classification in networked data: A toolkit and a univariate case study. *The Journal of Machine Learning Research*, 8:935–983, 2007.

[26] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[27] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119. 2013.

[28] T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of NAACL-HLT*, pages 746–751, 2013.

[29] A. Mnih and G. E. Hinton. A scalable hierarchical distributed language model. *Advances in neural information processing systems*, 21:1081–1088, 2009.

[30] F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pages 246–252, 2005.

[31] J. Neville and D. Jensen. Iterative classification in relational data. In *Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 13–20, 2000.

[32] J. Neville and D. Jensen. Leveraging relational autocorrelation with latent group models. In *Proceedings of the 4th International Workshop on Multi-relational Mining*, MRDM '05, pages 49–55, New York, NY, USA, 2005. ACM.

[33] J. Neville and D. Jensen. A bias/variance decomposition for models using collective inference. *Machine Learning*, 73(1):87–106, 2008.

[34] M. E. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[35] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[36] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems 24*, pages 693–701. 2011.

[37] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

[38] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90. ACM, 2004.

[39] L. Tang and H. Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 817–826, New York, NY, USA, 2009. ACM.

[40] L. Tang and H. Liu. Scalable learning of collective behavior based on sparse social dimensions. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1107–1116. ACM, 2009.

[41] L. Tang and H. Liu. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478, 2011.

[42] S. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *The Journal of Machine Learning Research*, 99:1201–1242, 2010.

[43] X. Wang and G. Sukthankar. Multi-label relational neighbor classification using social context features. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 464–472. ACM, 2013.

[44] W. Zachary. An information flow modelfor conflict and fission in small groups1. *Journal of anthropological research*, 33(4):452–473, 1977.