# Scalable Methods for Graph-Based Unsupervised and Semi-Supervised Learning

Frank Lin

Language Technologies Institute

School of Computer Science

Carnegie Mellon University

Committee

William W. Cohen (Chair)

Christos Faloutsos

Tom Mitchell

Xiaojin Zhu

July 2012

# Abstract

Data often comes in the form of a graph. When it does not, it often makes sense to represent it as a graph for learning tasks that rely on the similarities or relationships between data points. As data size grows, traditional methods for learning on graphs often become computationally intractable in terms of time and space requirements.

We describe new methods for graph-based clustering and semi-supervised classification with a focus on scalability. We show how these methods can be efficiently extended to work on non-graph data, and we demonstrate their application and effectiveness on a wide variety of datasets that includes social and citation networks, political blogs, document collections, noun phrase-context co-occurrences, and geolocations.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

Graph data is ubiquitous. Many types of data come in the form of a graph or a network, with nodes representing data points and edges between nodes representing relationships between data points. Examples include social networks, relational databases, router networks, and the World Wide Web—a network of interconnected web pages.

Some types of data do not come naturally in the form of a graph. Yet as the size of these types of data increases, it makes sense to *represent them as graphs* for the purposes of automated learning and discovery. This is true for at least two general tasks: clustering and semi-supervised learning.

First, for discovering hidden structure in data, we want to group similar data points together, but we do not know what the groups are. This is called *clustering*, a type of unsupervised learning. Graphs are natural data representations for this task because grouping is based on similarity, and it is natural to represent similarities between data points as weighted edges between nodes.

In the second case, we know what the groups are but we only know the group labels of a few of the data points. To infer the group labels for the rest of the data points,

we can leverage the similarity between data points and propagate labels from labeled to unlabeled data points. This is a type of *semi-supervised learning*. For label propagation, graphs are again natural representations of the similarities between data points.

As the amount of data increases, it becomes crucial that automated methods for clustering and semi-supervised learning be *scalable*—that is, they should be efficient space-wise and time-wise with respect to the number of data points, and the result they provide should be of the same quality regardless of data size. Additionally, the methods should be easily implemented in a computing framework which supports large-scale data, such as the MapReduce programming model of the Apache Hadoop project.

## 1.2 Thesis Goal

The goal of this thesis work is to make contributions toward fast, space-efficient, effective, and simple graph-based learning methods that scale up to large datasets. Specifically, we propose general unsupervised and semi-supervised methods that work on both natural network data and network data derived from pair-wise similarity between data points. We also propose to show these methods can be extended to efficiently solve a variety of interesting and difficult problems found in different types of datasets.

## 1.3 Contributions

The contributions of this thesis are made along two related lines of work—one for unsupervised learning and one for semi-supervised learning. Each line begins with a general method as its basic component, laying a groundwork for further modifications and extensions.

For unsupervised learning, we proposed *power iteration clustering* (PIC) as a general graph clustering method and a scalable alternative to spectral clustering (Chapter 2 and

3). For semi-supervised learning, we proposed *MultiRankWalk* (MRW) as a general graph learning method for when there are only a few training instances (Chapter 4 and 5).

To extend these graph-based methods to work on general feature vector data, we proposed the idea of *implicit manifolds* (IM). IM is a tool for transforming an $O(n^2)$ algorithm on an $O(n^2)$ data manifold into an $O(n)$ algorithm that outputs the exactly same solution. IM is also a framework that specifies the class of similarity functions and algorithms under which this can be done, which includes PIC and MRW (Chapter 6).

IM also paves the way for further extensions to PIC and MRW. We extended PIC to provide efficient mixed membership clustering via edge clustering (Chapter 7), and we extended random walk learning methods (e.g., MRW) to data in non-linear continuous manifolds such as the Gaussian kernel Hilbert space (Chapter 8).

We performed experiments on various types of real datasets for all of the proposed methods to test their effectiveness and scalability on real-world problems.

## 1.4 Notation

Unless otherwise specified, the general use of symbols will follow Table 1.1:

| Example | Explanation |
|---|---|
| $A$ | An uppercase letter denotes a matrix |
| $A(i,j)$ | The element at i-th row and j-th column of $A$ |
| $A(i,:)$ | The i-th row of $A$ |
| $A(:,j)$ | The j-th column of $A$ |
| $\mathbf{a}$ | A boldface lowercase letter denotes a vector |
| $\mathbf{a}(i)$ | The i-th element of $\mathbf{a}$ |
| $\mathcal{A}$ | A script uppercase letter denotes a set |

Table 1.1: General symbol explanations.

We also define some specific symbols that we will use frequently throughout this work. Given a dataset $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ where $\mathbf{x}_i$ is the feature vector of the i-th instance, a *similarity function* $s(\mathbf{x}_i, \mathbf{x}_j)$ is a function where $s(\mathbf{x}_i, \mathbf{x}_j) = s(\mathbf{x}_j, \mathbf{x}_i)$ and $s \geq 0$ if $i \neq j$.

Following previous work [91], it is mathematically convenient to define $s = 0$ if $i = j$. An *affinity matrix* or a *similarity matrix* $A \in \mathcal{R}^{n \times n}$ is defined by $A(i,j) = s(\mathbf{x}_i, \mathbf{x}_j)$. The *degree matrix* $D$ associated with $A$ is a diagonal matrix with $D(i,i) = \sum_j A(i,j)$. We will view $A$ interchangeably as a matrix, and an undirected graph with nodes $\mathcal{X}$ and the edge from $\mathbf{x}_i$ to $\mathbf{x}_j$ weighted by $s(\mathbf{x}_i, \mathbf{x}_j)$.

A *row-normalized affinity matrix* $W$ is defined as $D^{-1}A$; such a matrix is also referred to as a *right-stochastic* or *row-stochastic matrix*. Respectively, a *column-normalized affinity matrix* $P$ is defined as $AD^{-1}$, and is also referred to as a *left-stochastic* or *column-stochastic matrix*. Note that $P = W^T$ and $W = P^T$.

We also view $W$ or $P$ as a *probability transition matrix* for Markov random walks on the graph $A$. If a vector $\boldsymbol{\nu}^t$ of size $n$ defines a probability distribution over the nodes in $A$ at time $t$, then $\boldsymbol{\nu}^{t+1} = P\boldsymbol{\nu}^t$ is the probability distribution after taking one step forward in the Markov chain; conversely, $\boldsymbol{\nu}^{t+1} = W\boldsymbol{\nu}^t$ is the probability distribution after taking one step in reverse [73].

The above definitions are summarized in Table 1.2.

| Symbol | Definition |
|---|---|
| $\mathcal{X}$ | The dataset |
| $X$ | The feature matrix of $\mathcal{X}$; rows are instances and columns are features |
| $\mathbf{x}_i$ | The $i$-th instance of $\mathcal{X}$, also the $i$-th row of $X$ |
| $A$ | The affinity matrix of $\mathcal{X}$ |
| $D$ | The degree matrix (diagonal) |
| $I$ | The identity matrix (diagonal) |
| $W$ | The row-normalized affinity matrix: $W = D^{-1}A$ |
| $P$ | The column-normalized affinity matrix: $P = AD^{-1}$ |

Table 1.2: Specific symbol definitions.

# Chapter 2

# Power Iteration Clustering

## 2.1 Introduction

We proposed a simple and scalable graph-based clustering method called *power iteration clustering* (PIC) in [64]. In essence, PIC finds a very low-dimensional data embedding using truncated power iteration on a normalized pair-wise similarity matrix of the data points, and this embedding turns out to be an effective cluster indicator. Here we give a brief explanation of the method and experimental results.

PIC is related to a family of clustering methods called *spectral clustering*. PIC and spectral clustering are similar in that both embed data points in a low-dimensional subspace derived from the affinity matrix, and this embedding provides clustering results directly or through a k-means algorithm. They are different in what this embedding is and how it is derived. In spectral clustering the embedding is formed by the bottom eigenvectors of the Laplacian of an affinity matrix. In PIC the embedding is an approximation to a *eigenvalue-weighted linear combination* of *all* the eigenvectors of the row-normalized affinity matrix. This embedding turns out to be very effective for clustering, and in comparison to spectral clustering, the cost (in space and time) of explicitly

calculating eigenvectors is replaced by that of a small number of matrix-vector multiplications.

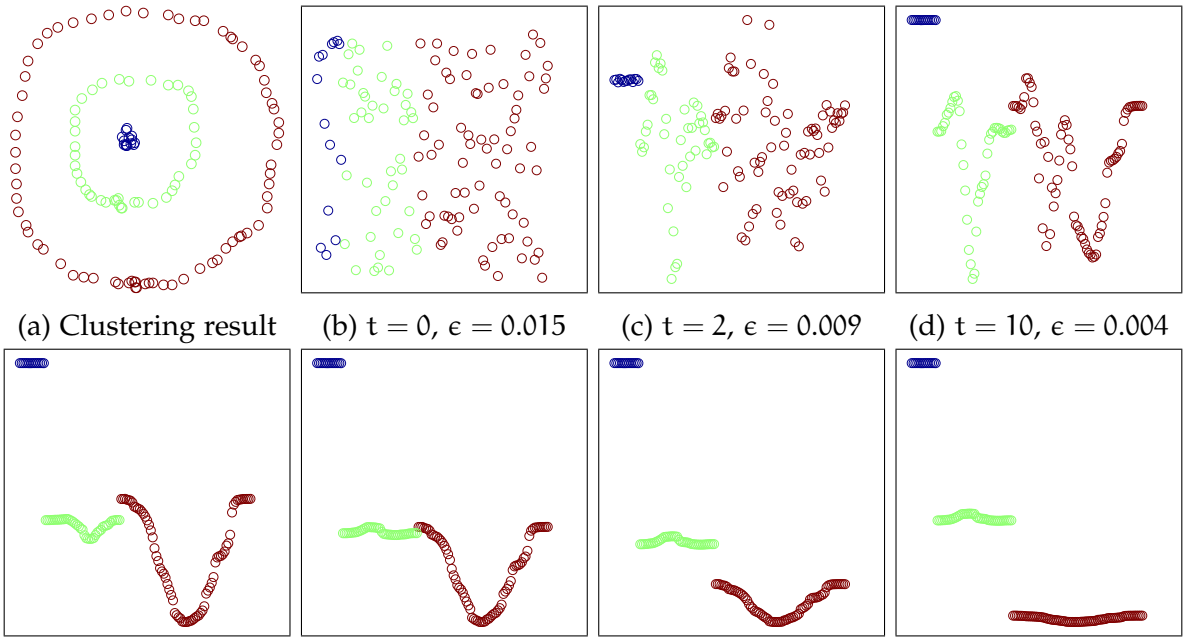
We tested PIC on a number of different types of datasets and obtain comparable or better clusters than existing spectral methods. However, the greatest advantage of PIC is its simplicity and scalability — we demonstrate that a basic implementation of this method is able to partition a network dataset of 100 million edges within a few seconds on a single machine, without sampling, grouping, or other preprocessing of the data.

## 2.2 Spectral Clustering

The row-normalized matrix $W$ of the affinity matrix $A$ derived from the data is closely related to the *normalized random-walk Laplacian* matrix $L$ of Meilă and Shi [73], defined as $L = I - D^{-1}A$. The second-smallest eigenvector of $L$ (the eigenvector with the second-smallest eigenvalue) defines a partition of the graph $W$ that approximately maximizes the *Normalized Cut* criteria. More generally, the $k$ smallest eigenvectors define a subspace where the clusters are often well-separated. Thus the second-smallest, third-smallest, ..., $k^{th}$ smallest eigenvectors of $L$ are often well-suited for clustering the graph $W$ into $k$ components [73], as in Algorithm 1. Most of spectral clustering algorithms have a similar structure, and mainly differ in Step 2 and Step 4.

**Algorithm 1** k-way Normalized Cuts [73]

1: **procedure** NCut($A, k$)
2: $W \leftarrow I - D^{-1}A$
3: Find eigenvalues $\lambda_1 \ldots \lambda_n$ of $W$ and corresponding eigenvectors $\boldsymbol{e}_1 \ldots \boldsymbol{e}_n$ where $\lambda_i$ is the $i$-th smallest eigenvalue of $W$
4: Let columns of $E$ be $\boldsymbol{e}_2 \ldots \boldsymbol{e}_k$
5: Use k-means on rows of $E$ to get cluster assignments $C_1, C_2, ..., C_k$.
6: **return** $C_1, C_2, ..., C_k$
7: **end procedure**



(a) Clustering result (b) $t = 0$, $\epsilon = 0.015$ (c) $t = 2$, $\epsilon = 0.009$ (d) $t = 10$, $\epsilon = 0.004$

(e) $t = 100$, $\epsilon = 0.003$ (f) $t = 200$, $\epsilon = 0.002$ (g) $t = 500$, $\epsilon = 0.002$ (h) $t = 1000$, $\epsilon = 0.001$

Figure 2.1: Clustering result and the embedding provided by $v^t$ for the 3Circles dataset. In (b) through (h), the value of each component of $v^t$ is plotted against its index, and the colors indicate cluster membership. Plots (b) through (h) are re-scaled so the largest value is always at the very top and the minimum value at the very bottom, and $\epsilon$ is the scale — maximum value minus the minimum value.

## 2.3 Power Iteration

The $k$ *smallest* eigenvectors of $L$ are also the $k$ *largest* eigenvectors of $W$. One simple method for computing the largest eigenvector of a matrix is *power iteration* (PI), also called the *power method*. PI is an iterative method, which starts with an arbitrary vector $v^0 \neq 0$ and repeatedly performs the update

$$v^{t+1} = cWv^t$$

where $c$ is a normalizing constant to keep $v^t$ from getting too large (here $c = 1/\left\|Wv^t\right\|_1$). While running PI *to convergence* on $W$ does not lead to an interesting result, the intermediate vectors obtained by PI during the convergence process are extremely interesting. This is best illustrated by example. Figure 2.1a shows a simple dataset—i.e., each $x_i$ is a

point in $\mathcal{R}^2$ space, with $s(x_i, x_j)$ defined as $exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma^2}\right)$. Figures 2.1b to 2.1h shows $v^t$ at various values of $t$, each illustrated by plotting $v^t(i)$ for each $x_i$. For purposes of visualization, the instances $x$ in the "bulls-eye" (blue) are ordered first, followed by instances in the central ring (green), then by those in the outer ring (red). As we can see the differences between the distinct values of the $v^t$'s become smaller as $t$ increases. Qualitatively, PI first converges *locally* within a cluster: by $t = 400$ the points from each cluster have approximately the same value in $v^t$, leading to three disjoint line segments in the visualization. Then, after local convergence, the line segments draw closer together more slowly.

## 2.4 PI Convergence Analysis

Let us assume that $W$ has eigenvectors $e_1, \ldots, e_n$ with eigenvalues $\lambda_1, \ldots, \lambda_n$, where $\lambda_1 = 1$ and $e_1$ is constant. Given $W$, we define the *spectral representation* of a value $a \in \{1, \ldots, n\}$ to be the vector $s_a = \langle e_1(a), \ldots, e_k(a) \rangle$, and define the *spectral distance between* $a$ *and* $b$ as

$$spec(a, b) \equiv \|s_a - s_b\|_2 = \sqrt{\sum_{i=2}^{k} (e_i(a) - e_i(b))^2}$$

Usually in spectral clustering it is assumed that the eigenvalues $\lambda_2, \ldots, \lambda_k$ are larger than the remaining ones. We define $W$ to have an $(\alpha, \beta)$-*eigengap between the* $k$-*th and* $(k+1)$-*th eigenvector* if $\lambda_k/\lambda_2 \geq \alpha$ and $\lambda_{k+1}/\lambda_2 \leq \beta$. We will also say that $W$ is $\gamma_e$-*bounded* if $\forall i, a, b \in \{1, \ldots, n\}$, $|e_i(a) - e_i(b)| \leq \gamma_e$; note that every $W$ is $\gamma_e$-bounded for some $\gamma_e$. Letting $v^t$ be the result of the $t$-th iteration of PI, we define the $(t, v^0)$-*distance between* $a$ *and* $b$ as

$$pic^t(v^0; a, b) \equiv |v^t(a) - v^t(b)|$$

For brevity, we will usually drop $\boldsymbol{v}^0$ from our notation (e.g., writing $\mathrm{pic}^t(a,b)$). Our goal is to relate $\mathrm{pic}^t(a,b)$ and $\mathrm{spec}(a,b)$. Let us first define

$$\mathrm{signal}^t(a,b) \equiv \sum_{i=2}^{k} [\boldsymbol{e}_i(a) - \boldsymbol{e}_i(b)] c_i \lambda_i^t$$
$$\mathrm{noise}^t(a,b) \equiv \sum_{j=k+1}^{n} [\boldsymbol{e}_j(a) - \boldsymbol{e}_j(b)] c_j \lambda_j^t$$

**Proposition 1.** *For any $W$ with $\boldsymbol{e}_1$ a constant vector,*

$$\mathrm{pic}^t(a,b) = |\mathrm{signal}^t(a,b) + \mathrm{noise}^t(a,b)|$$

*Proof.* To verify the proposition, note that (ignoring renormalization)

$$\begin{aligned}\boldsymbol{v}^t &= W\boldsymbol{v}^{t-1} = W^2\boldsymbol{v}^{t-2} = \ldots = W^t\boldsymbol{v}^0 \\ &= c_1 W^t \boldsymbol{e}_1 + c_2 W^t \boldsymbol{e}_2 + \ldots + c_n W^t \boldsymbol{e}_n \\ &= c_1 \lambda_1^t \boldsymbol{e}_1 + c_2 \lambda_2^t \boldsymbol{e}_2 + \ldots + c_n \lambda_n^t \boldsymbol{e}_n\end{aligned}$$

Rearranging terms,

$$\begin{aligned}\mathrm{pic}^t(a,b) = \Bigg| &[\boldsymbol{e}_1(a) - \boldsymbol{e}_1(b)] c_1 \lambda_1^t \\ &+ \sum_{i=2}^{k} [\boldsymbol{e}_i(a) - \boldsymbol{e}_i(b)] c_i \lambda_i^t + \sum_{j=k+1}^{n} [\boldsymbol{e}_j(a) - \boldsymbol{e}_j(b)] c_j \lambda_j^t \Bigg|\end{aligned}$$

where the second and third terms correspond to $\mathrm{signal}^t$ and $\mathrm{noise}^t$ respectively, and the first term is zero because $\boldsymbol{e}_1$ is a constant vector. □

The implications of the proposition are somewhat clearer if we define a "radius" $R^t \equiv \frac{1}{\lambda_2^t}$ and consider the product of $R^t$ and the quantities above:

$$R^t signal^t(a,b) = \sum_{i=2}^{k} [\boldsymbol{e}_i(a) - \boldsymbol{e}_i(b)] c_i \left(\frac{\lambda_i}{\lambda_2}\right)^t \tag{2.1}$$

$$R^t noise^t(a,b) \leq \sum_{j=k+1}^{n} \gamma_e c_j \beta^t \tag{2.2}$$

After rescaling points by $R^t$, we see that $noise^t$ will shrink quickly, if the $\beta$ parameter of the eigengap is small. We also see that $signal^t$ is an approximate version of $spec$: the differences are that $signal^t$:

1. is compressed to the small radius $R^t$,
2. has components distorted by $c_i$ and $(\lambda_i/\lambda_2)^t$, and
3. has terms that are additively combined (rather than combined with Euclidean distance).

Note that the size of the radius is of no importance in clustering, since most clustering methods (e.g., k-means) are based on the relative distance between points, not the absolute distance. Furthermore, if the $c_i$'s are not too large or too small, the distorting factors are dominated by the factors of $(\lambda_i/\lambda_2)^t$, which implies that the importance of the dimension associated with the $i$-th eigenvector is down-weighted by (a power of) its eigenvalue; in Section 2.6 we will show that experimentally, this weighting scheme often *improves* performance for spectral methods.

We are then left with the difference 3 that the terms in the sum defining $signal^t$ are additively combined. How does this effect the utility of $signal^t$ as a cluster indicator? In prior work by Meilă and Shi [73], it is noted that for many natural problems, $W$ is approximately block-stochastic, and hence the first $k$ eigenvectors are approximately piecewise constant over the $k$ clusters. This means that if $a$ and $b$ are in the same

cluster, $spec(a, b)$ would be nearly 0, and conversely if $a$ and $b$ are in different clusters, $spec(a, b)$ would be large.

It is easy to see that when $spec(a, b)$ is small, $signal^t$ must also be small. However, when $a$ and $b$ are in different clusters, since the terms are signed and additively combined, it is possible that they may "cancel each other out" and make $a$ and $b$ seem to be of the same cluster. Fortunately, this seems to be uncommon in practice when the number of clusters $k$ is not too large. Hence, for large enough $\alpha$, small enough $t$, $signal^t$ is very likely a good cluster indicator.

## 2.5 PI Truncated

These observations suggest that an effective clustering algorithm might run PI for some small number of iterations $t$, stopping *after* it has converged within clusters but *before* final convergence, leading to an approximately piecewise constant vector, where the elements that are in the same cluster have similar values. Specifically, define the *velocity at* $t$ to be the vector $\delta^t = v^t - v^{t-1}$ and define the *acceleration at* $t$ to be the vector $\epsilon^t = \delta^t - \delta^{t-1}$. We pick a small threshold $\hat{\epsilon}$ and stop PI when $\left\|\epsilon^t\right\|_\infty \leq \hat{\epsilon}$.

Our stopping heuristic is based on the assumption and observation that while the clusters are "locally converging", the rate of convergence changes rapidly; whereas during the final global convergence, the convergence rate appears more stable. This assumption turns out to be well-justified. Recall that $v^t = c_1\lambda_1^t e_1 + c_2\lambda_2^t e_2 + ... + c_n\lambda_n^t e_n$. Then

$$\frac{v^t}{c_1\lambda_1^t} = e_1 + \frac{c_2}{c_1}\left(\frac{\lambda_2}{\lambda_1}\right)^t e_2 + ... + \frac{c_n}{c_1}\left(\frac{\lambda_n}{\lambda_1}\right)^t e_n$$

It can be seen that the convergence rate of PI toward the dominant eigenvector $e_1$ depends on $(\lambda_i/\lambda_1)^t$ for the significant terms $i = 2, ..., k$, since their eigenvalues are close to 1 if the clusters are well-separated [73], making $(\lambda_i/\lambda_1)^t \simeq 1$. This implies that in the

beginning of PI, it is converging to a linear combination of the top $k$ eigenvectors, with terms $k+1, \ldots, n$ diminishing at a rate of $\geq (\lambda_{k+1}/1)^t$. After the noise terms $k+1, \ldots, n$ go away, the convergence rate toward $\boldsymbol{e}_1$ becomes nearly constant. We call the complete method *power iteration clustering* (PIC), described in Algorithm 2. In all experiments in this chapter, we let $\hat{\epsilon} = \frac{1 \times 10^{-5}}{n}$ where $n$ is the number of data instances.

**Algorithm 2** The PIC algorithm

**procedure** PIC($W$, $k$)
    Initialize $\boldsymbol{v}^0$
    **repeat**
        $\boldsymbol{v}^{t+1} \leftarrow \frac{W\boldsymbol{v}^t}{\|W\boldsymbol{v}^t\|_1}$
        $\delta^{t+1} \leftarrow |\boldsymbol{v}^{t+1} - \boldsymbol{v}^t|$
        $t \leftarrow t+1$
    **until** $|\delta^t - \delta^{t-1}| \simeq 0$
    Use k-means on elements of $\boldsymbol{v}^t$ to get cluster assignments $C_1, C_2, ..., C_k$.
    **return** $C_1, C_2, ..., C_k$
**end procedure**

## 2.6 Experiments

### 2.6.1 Datasets

We demonstrate the effectiveness of PIC on a variety of real datasets; they have known labels and have been used for both classification and clustering tasks:

- Iris contains flower petal and sepal measurements from three species of irises, two of which are non-linearly separable [37].
- PenDigits01 and PenDigits17 are hand-written digit datasets [7] with digits "0", "1" and "1", "7", respectively. Each dataset contains 200 instances, 100 per digit. PenDigits01 represents an "easy" dataset and PenDigits17 a "difficult" dataset.
- PolBooks is co-purchase network of political books [79]. Each book is labeled "liberal", "conservative", or "neutral", with most belonging to the first two category.

- UBMCBlog is a connected network dataset of liberal and conservative political blogs mined from blog posts [51].
- AGBlog is a connected network dataset of liberal and conservative political blogs mined from blog homepages [5].
- 20ng* datasets are subsets of the 20 newsgroups text dataset [75]. 20ngA contains 100 documents from 2 newsgroups: *misc.forsale* and *soc.religion.christian*. 20ngB adds 100 documents to each group of 20ngA. 20ngC adds 200 from *talk.politics.guns* to 20ngB. 20ngD adds 200 from *rec.sport.baseball* to 20ngC.

For the network datasets (Polbooks, UBMGBlog, AGBlog), the affinity matrix is simply $A_{ij} = 1$ if blog $i$ has a link to $j$ or vice versa, otherwise $A_{ij} = 0$. For all other datasets, the affinity matrix is simply the cosine similarity between feature vectors: $\frac{\mathbf{x}_i \cdot \mathbf{x}_j}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2}$. Cosine similarity is used instead of the distance function used in Figure 2.1 to avoid having to tune $\sigma^2$. For the text datasets, word counts are used as feature vectors with only stop words and singleton words removed. Table 2.1 contains a summary of dataset size information.

| Dataset | Nodes | Edges | k |
|---|---|---|---|
| Iris | 150 | * | 3 |
| PenDigits01 | 200 | * | 2 |
| PenDigits17 | 200 | * | 2 |
| PolBooks | 105 | 441 | 3 |
| UBMCBlog | 404 | 2382 | 2 |
| AGBlog | 1222 | 16714 | 2 |
| 20ngA | 200 | * | 2 |
| 20ngB | 400 | * | 2 |
| 20ngC | 600 | * | 3 |
| 20ngD | 800 | * | 4 |

Table 2.1: Sizes of datasets for clustering. *Nodes* are the number of nodes in the graph-view of the dataset, which corresponds to the number of instances in the dataset. *Edges* are the number of edges, an * indicates it is the number of nodes squared (a complete graph).

| | | NCut | | | NJW | | | PIC | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | k | Purity | NMI | RI | Purity | NMI | RI | Purity | NMI | RI |
| Iris | 3 | 0.67 | 0.72 | 0.78 | 0.77 | 0.61 | 0.80 | **0.98** | **0.93** | **0.97** |
| PenDigits01 | 2 | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** |
| PenDigits17 | 2 | **0.76** | **0.21** | **0.63** | **0.76** | 0.20 | **0.63** | **0.76** | **0.21** | **0.63** |
| PolBooks | 3 | 0.85 | 0.57 | 0.84 | 0.83 | 0.54 | 0.83 | **0.87** | **0.62** | **0.86** |
| UBMCBlog | 2 | **0.95** | **0.75** | **0.91** | **0.95** | 0.74 | **0.91** | 0.95 | 0.72 | **0.90** |
| AGBlog | 2 | 0.52 | 0.01 | 0.50 | 0.52 | 0.00 | 0.50 | **0.96** | **0.75** | **0.92** |
| 20ngA | 2 | **0.96** | **0.76** | **0.92** | **0.96** | **0.76** | **0.92** | **0.96** | **0.76** | **0.92** |
| 20ngB | 2 | 0.51 | 0.01 | 0.50 | 0.55 | 0.08 | 0.51 | **0.87** | **0.52** | **0.77** |
| 20ngC | 3 | 0.62 | 0.33 | 0.67 | 0.63 | 0.35 | 0.69 | **0.69** | **0.44** | **0.74** |
| 20ngD | 4 | 0.48 | 0.24 | 0.63 | 0.52 | 0.30 | 0.68 | **0.58** | **0.31** | **0.71** |
| Average | | 0.73 | 0.46 | 0.74 | 0.75 | 0.46 | 0.75 | **0.86** | **0.63** | **0.84** |

Table 2.2: Clustering performance of PIC and spectral clustering algorithms on several real datasets. For all measures a higher number means better clustering. Bold numbers are the highest in its row.

### 2.6.2 Accuracy Results

We evaluate the clustering results against the labels using three measures: *cluster purity* (Purity), *normalized mutual information* (NMI), and *Rand index* (RI). Three measures are used to ensure a thorough evaluation of clustering results; for example, NMI takes into account the difference between the number of clusters returned by the method and the number of classes according to the ground truth labels, which is ignored by Purity. See Appendix A.2 for formal definitions of these measures.

We also compare the results of PIC against those of spectral clustering methods Normalized Cuts (NCut) [73, 91] and the Ng-Jordan-Weiss algorithm (NJW) [80]. The NJW algorithm is very similar to that of NCut as in Algorithm 1, with two differences being the Laplacian matrix is defined as $W = I - D^{1/2}AD^{1/2}$ in Step 2 and eigenvectors $e_1 \dots e_k$ are all used in Step 4. The clustering results comparing PIC against these methods are shown in Table 2.2.

On most datasets PIC does equally well or does better than the other methods for most evaluation measures. In the cases where NCut or NJW fails badly (AGBlog, 20ngB),

the most likely cause is that the top $k$ eigenvectors of the graph Laplacian fail to provide a good low-dimensional embedding for $k$-means. This problem may be alleviated by use of additional heuristics to choose the "good" eigenvectors and discard the "bad" eigenvectors [63, 105, 109]. PIC, on the other hand, does not pick eigenvectors — the embedding uses a weighted linear combinations of *all* the eigenvectors.

### 2.6.3 Eigenvalue Weighting

As seen in Section 2.4, in distance metric used by PIC, the $i$-th eigenvector is weighted by $c_i\lambda_i^t$, with $\lambda^t$ dominating the weight term as the number of iteration $t$ grows. In other words, the eigenvectors are weighted according to their corresponding eigenvalues raised to the power $t$. Based on analyses in [73, 103], weighting the eigenvectors according to eigenvalues seems reasonable, since eigenvalues of a row-normalized affinity matrix range from 0 to 1 and good cluster indicators should have eigenvalues close to 1 and "spurious" eigenvalues close to 0 (the opposite is true in the case of normalized Laplacian matrices). To test this on real data, we run the NCut algorithm with the following modification: instead of using the first $k$ eigenvectors, we use the first 10 vectors and weight them in different ways. First, we use them directly without any additional weighting. Second, we scale them by their corresponding eigenvalues. Lastly, we scale them by their corresponding eigenvalue raised to a power $t$. The result is shown in Table 2.3.

In Table 2.3, we see that using the eigenvectors with uniform weights produces poor clustering on most datasets, even on PenDigits01, which is a relatively easy dataset. However, note that it does rather well on the blog datasets, and it even outperforms the original NCut algorithm on AGBlog, showing that original NCut is missing an important cluster indicator by choosing only the first $k$ eigenvectors. In this light, we can also view PIC as an approximation to the eigenvalue-weighted modification of NCut.

| | | uniform weights | | | $e_i$ weighted by $\lambda_i$ | | | $e_i$ weighted by $\lambda_i^{15}$ | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | k | Purity | NMI | RI | Purity | NMI | RI | Purity | NMI | RI |
| Iris | 3 | 0.67 | 0.65 | 0.73 | **0.98** | **0.93** | **0.97** | **0.98** | **0.93** | **0.97** |
| PenDigits01 | 2 | 0.70 | 0.28 | 0.58 | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** |
| PenDigits17 | 2 | 0.70 | 0.18 | 0.58 | **0.76** | **0.21** | **0.63** | **0.76** | **0.21** | **0.63** |
| PolBooks | 3 | 0.49 | 0.10 | 0.44 | **0.85** | 0.59 | **0.85** | 0.84 | **0.59** | 0.85 |
| UBMCBlog | 2 | 0.95 | 0.74 | 0.91 | 0.95 | 0.74 | 0.91 | **0.95** | **0.75** | **0.91** |
| AGBlog | 2 | **0.95** | 0.71 | 0.90 | **0.95** | **0.72** | **0.91** | **0.95** | **0.72** | **0.91** |
| 20ngA | 2 | 0.56 | 0.07 | 0.51 | **0.96** | **0.76** | **0.92** | 0.95 | 0.70 | 0.90 |
| 20ngB | 2 | 0.71 | 0.27 | 0.59 | **0.95** | **0.70** | **0.90** | 0.51 | 0.01 | 0.50 |
| 20ngC | 3 | 0.69 | 0.39 | 0.66 | **0.66** | 0.38 | **0.70** | 0.64 | **0.47** | 0.68 |
| 20ngD | 4 | 0.48 | 0.24 | 0.64 | 0.49 | 0.26 | 0.64 | **0.53** | **0.29** | **0.71** |
| Average | | 0.69 | 0.36 | 0.65 | **0.85** | **0.63** | **0.84** | 0.81 | 0.57 | 0.81 |

Table 2.3: Clustering performance of eigenvalue-weighted NCut on several real datasets. For all measures a higher number means better clustering. Boldface numbers are the highest in its row.

### 2.6.4 Scalability Results

Perhaps one of the greatest advantages of PIC lies in its scalability. Space-wise it needs only a single vector of size $n$ for $v^t$ and two more of the same dimension to keep track of convergence. Speed-wise, power iteration is known to be fast on sparse matrices and converges fast on many real-world datasets; yet PIC converges *even more quickly* than power iteration, since it naturally stops when $v^t$ is no longer accelerating towards convergence. Table 2.4 shows the runtime of PIC and the spectral clustering algorithms on datasets described in the previous section, and Table 2.5 shows the runtime on large, synthetic datasets.

For testing the runtime of the spectral clustering algorithms, we tried two different ways of finding eigenvectors. $\text{NCut}_{\text{EVD}}$ uses the slower, classic eigenvalue decomposition method to find all the eigenvectors. $\text{NCut}_{\text{IRAM}}$ uses the fast *Implicitly Restarted Arnoldi Method* (IRAM) [60], a memory-efficient version of the fast Lanczos algorithm for non-symmetric matrices. With IRAM we can ask only for the top k eigenvectors, resulting in less computation time.

We also tested PIC on synthetic datasets generated with a modified version of the Erdős-Rényi random network model, which generates a connected block-stochastic graph with two components. The $n$ nodes are in two equal-sized blocks, and the number of edges is $0.01n^2$. We define an additional parameter $e = 0.2$, and when generating a edge between two nodes, with probability $1 - e$ the edge is placed between two random nodes of the same cluster and otherwise between two nodes of different clusters. Such models for generating synthetic networks are also called *planted partition models* [24]. Results are averaged over five random network datasets and three trials for each dataset, and all accuracies are above 0.99. As expected from the analysis in Section 2.5, the number of iterations PIC requires does not increase with dataset size; real datasets average 13 iterations and the largest synthetic dataset converges in 3 iterations. $\text{NCut}_{\text{EVD}}$ and $\text{NCut}_{\text{IRAM}}$ were not run on the largest synthetic datasets because they required more memory than was available. [1]

| Dataset | Nodes | $\text{NCut}_{\text{EVD}}$ | $\text{NCut}_{\text{IRAM}}$ | PIC |
|---|---|---|---|---|
| Iris | 150 | 17 | 61 | 1 |
| PenDigits01 | 200 | 28 | 23 | 1 |
| PenDigits17 | 200 | 30 | 36 | 1 |
| PolBooks | 102 | 7 | 22 | 1 |
| UBMCBlog | 404 | 104 | 32 | 1 |
| AGBlog | 1,222 | 1095 | 70 | 3 |
| 20ngA | 200 | 32 | 37 | 1 |
| 20ngB | 400 | 124 | 56 | 3 |
| 20ngC | 600 | 348 | 213 | 5 |
| 20ngD | 800 | 584 | 385 | 10 |

Table 2.4: Runtime comparison (in milliseconds) of PIC and spectral clustering algorithms on several real datasets.

[1] Implemented in MATLAB and run on a Linux machine with two quad-core 2.26Ghz CPUs and 24GB RAM.

| Nodes | Edges | $NCut_{EVD}$ | $NCut_{IRAM}$ | PIC |
|---|---|---|---|---|
| 1k | 10k | 1,885 | 177 | 1 |
| 5k | 250k | 154,797 | 6,939 | 7 |
| 10k | 1,000k | 1,111,441 | 42,045 | 34 |
| 50k | 25,000k | - | - | 849 |
| 100k | 100,000k | - | - | 2,960 |

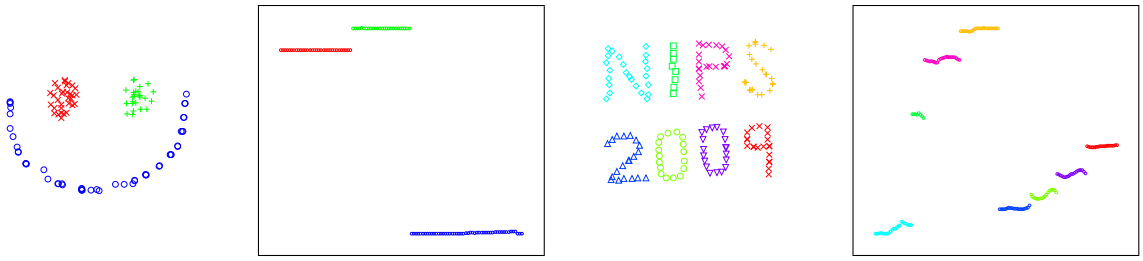
Table 2.5: Runtime comparison (in milliseconds) of PIC and spectral clustering algorithms on synthetic datasets.

## 2.7 Multi-dimensional PIC Embedding

The basic PIC algorithm in Section 2 projects the data points onto a one-dimensional embedding. It is conceivable that when the number of clusters $k$ increases, so does the likelihood of collision of two clusters "colliding" on a single dimension. A collision can happen because the underlying PIC embedding value of a cluster is determined by the initial vector $\boldsymbol{v}^0$, which is chosen arbitrarily. If $k$ is sufficiently large, then it becomes likely that the embedding values of two clusters are assigned to be "close" to each other, and due to the presence of noise in data, the two clusters may become difficult to distinguish in the resulting embedding $\boldsymbol{v}^t$.

This is illustrated in Figure 2.2, where two synthetic datasets, "Smiley" (Figure 2.2a) and "NIPS 2009" (Figure 2.2c), are clustered using PIC, with their respective one-dimensional PIC embedding plots (Figure 2.2b and 2.2d). Although PIC is able to cluster these correctly, embedding for some of the clusters in Figure 2.2d are very close and can potentially be determined to be of the same cluster by $k$-means.

One way to avoid such "collisions" is to set the initial vector such that nodes in different clusters are likely to have different initial values; for instance, letting the initial vector map to a function of its corresponding node's degree. Another way to avoid collisions is to run power iteration with early stopping $d$ times ($d \ll k$) and embed the nodes in the $d$-dimensional space spanned by these vectors. To evaluate their effectiveness we choose three variations of PIC for comparison: $PIC_R$ (one random initial vector), $PIC_D$



(a) "Smiley" (b) $\boldsymbol{v}^t$ for "Smiley" (c) "NIPS 2009" (d) $\boldsymbol{v}^t$ for "NIPS 2009"

Figure 2.2: Toy datasets and the corresponding PIC embedding provided by $\boldsymbol{v}^t$. The embedding is plotted in the same way as in Figure 2.1.

(one initial vector set according to the diagonal of D), and $\text{PIC}_{\text{R4}}$ (four random initial vectors) [12]. These variations are compared against $\text{NCut}_{\text{EVD}}$ and $\text{NJW}_{\text{EVD}}$ (full eigen-decomposition implementations of these spectral clustering methods) on a number of synthetic and real network datasets [12]. The nodes in the graphs in all the datasets studied have labels, which are used only to evaluate the accuracy of clustering and not given as input to any of the clustering methods.

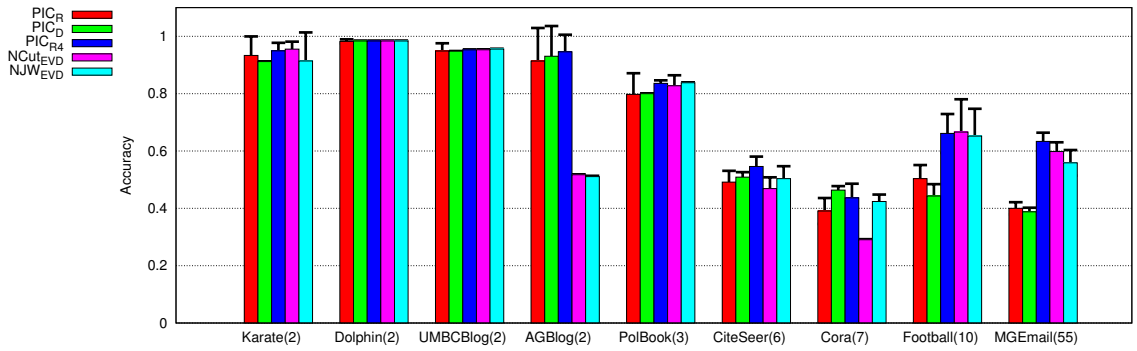
The first type of datasets consists of social networks, citation networks, and similar networks that have been studied in the sociology literature. The nodes in the PolBook dataset [57] are political books, and edges represent co-purchasing behavior. Books are labeled "liberal", "conservative", or "neutral", based on their viewpoint. The nodes in the Karate dataset [108] are members of a karate club, and the edges are friendships. The labels are sub-communities, as defined by two subgroups that formed after a breakup of the original community. The Dolphin dataset [70] is a similar social network of associations between dolphins in a pod in Doubtful Sound, New Zealand, and labels correspond to sub-community membership after a similar breakup. The nodes in the Football dataset are Division IA colleges in the United States, and the edges represent games in the 2000 regular season, and the labels represent regional conferences [41]. The nodes in the MGEmail corpus [74] are MBA students, organized in teams of four to six members, who ran simulated companies over a 14-week period as part of a management course at Carnegie Mellon University. The edges correspond to emails, and the true cluster

labels correspond to teams. In the UMBC [64], AG [5], and MSP [65] datasets, the nodes are blogs, and an edge between two nodes represents hyperlinks. Blog sites are labeled either liberal or conservative. The MSP dataset also contains news cites, which are unlabeled. In the Cora and CiteSeer datasets [69], nodes are scientific papers, and links are citations. The node labels are scientific subfields. The nodes in the Senate dataset are US Senators, and edges are agreement on congressional votes. The labels correspond to political party. Unlike other datasets, this is a complete graph.

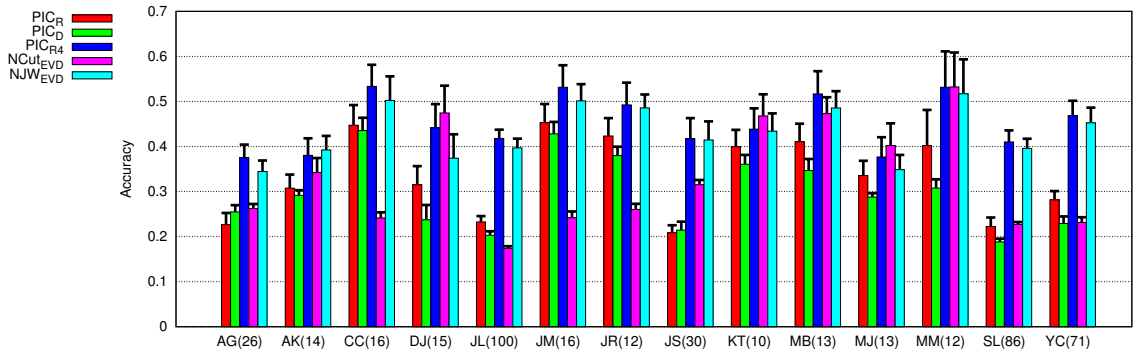
The second type of datasets were used for author disambiguation [43]. Each of the dataset corresponds to a first name initial and a common last name. The datasets were constructed by extracting co-authorship information for papers authored by people with these ambiguous first initial-last name pairs. In each dataset, there are two types of nodes: (a) one node for each distinct name string, and (b) one node for each occurrence of a name in the list of authors of a paper. Edges link a name occurrence with the corresponding name string, and also link the name occurrence nodes for co-authors of a paper. The label of name occurrence nodes correspond to the id of the person associated with this name occurrence, and name string nodes are unlabeled.

The results on the real datasets are shown in Figure 2.3, with the following observations:

1. One-dimensional PIC loses clustering accuracy as the number of true clusters increases (due to collisions).
2. One-dimensional PIC using the diagonal of D does not seem to improve clustering accuracy in general on these datasets.
3. Multi-dimensional PIC is able to do as well or better than NCut or NJW implemented using full eigendecomposition, even when the number of clusters is large.
4. Where generally spectral clustering methods are recommended to use number of eigenvectors equal to that of the number of clusters (the subspace dimensionality




(a) Clustering accuracy on 9 network datasets.




(b) Clustering accuracy on 14 author disambiguation datasets.

Figure 2.3: Clustering accuracy on two types of datasets. In parentheses is the number of clusters for that particular dataset. $\text{PIC}_\text{R}$ and $\text{PIC}_{\text{R4}}$ results are averaged over 20 embedding trials and 50 k-means trials per embedding; $\text{PIC}_\text{D}$, NCut, and NJW results are averaged over 50 k-means trials.

equal to k), multi-dimensional PIC require a much smaller number of dimensions to do just as well or better as evident in Figure 2.3b.

To further illustrate how multi-dimensional PIC embedding works to avoid cluster collision, we plot each college in the Football dataset according to a two-dimensional PIC embedding (i.e., two one-dimensional PIC embedding) in Figure 2.4. The colors represent the conferences and the histograms corresponding to the axis on which they lie show what the embedding would be if only one of the dimensions were used. We can see that some of the clusters that "collide" in a one-dimensional embedding is better separated in two dimensions.

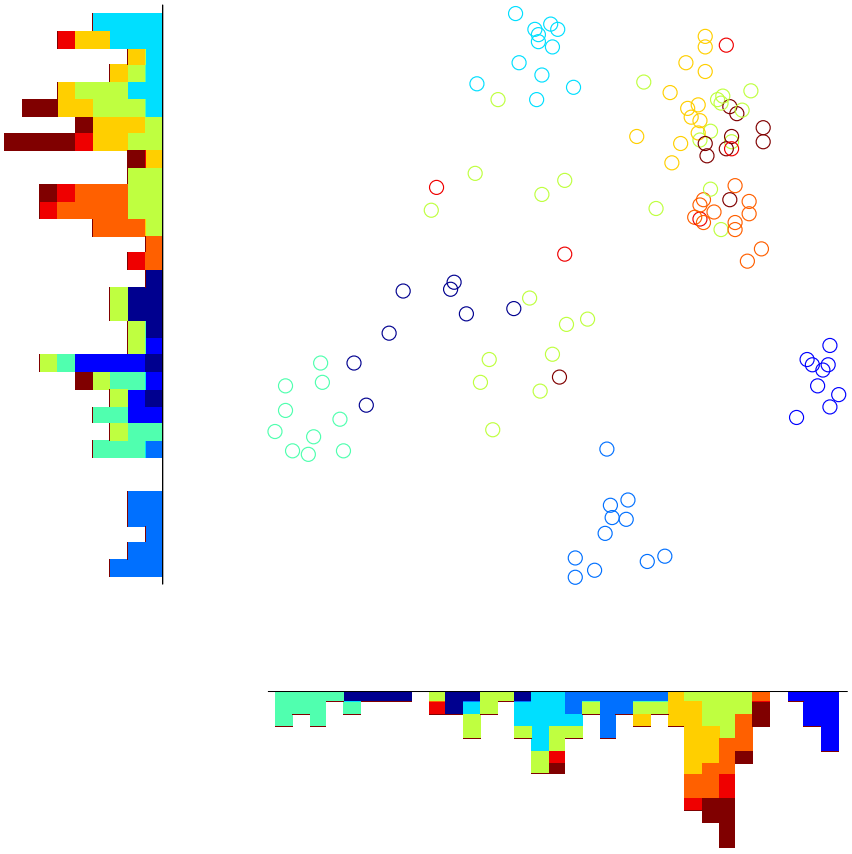


Figure 2.4: A visualization of the Football dataset using a 2-dimensional PIC embedding. Each point in the center scatter plot is a Division IA college, its x and y position determined by two independent one-dimensional PIC embeddings, and the color represent to the college conference it belongs to. The histograms show what the embedding would be if only one of the dimensions were used.

Multi-dimensional PIC shows substantial improvement over one-dimensional PIC when the number of clusters $k$ is large, but how do we determine the number of dimensions $d$? As a general strategy we recommend setting $d$ according to a logarithmic function of $k$ (e.g., $d = \lceil \log k \rceil$). This is an intuitive strategy; if a line of length $n$ provides just enough space so that $k$ random and noisy clusters can be embedded with little probability of collision, then a $d$-dimensional space will provide $n^d$ amount of space in which to "fit" more of these clusters. Formal results can be obtained via PIC embedding's connection to *diffusion maps* and *random projection* (See Chapter 3 and Section 3.6 for details of the connection). For example, if we assume that the clusters in the space

spanned by the $\lambda^t$-weighted eigenvectors of $W$ are distributed according to $k$ mixtures of Gaussians, then the data can be projected onto just $O(\log k)$ dimensions while still retaining the approximate level of separation between the clusters [26].

# Chapter 3

# PIC's Relation to Other Methods

Power iteration clustering (PIC) is a algorithmically very simple; however, the method, especially the embedding on which does clustering, can be viewed in many ways. In order to better understand how it works, how its different parameters affect the result, and how to improve, extend, and apply the method to different tasks, it is important to understand its connection to many of the existing methods and ideas. To that end we devote a chapter to PIC's relation to other methods.

## 3.1 Spectral Clustering

Spectral clustering began with the discovery of the correlation between the eigenvalues of the Laplacian matrix and the connectivity of a graph [36]. Much later it was introduced to the machine learning community through Ratio Cut [88] and Normalized Cuts [91], and it since has sparked much interest and lead to further modifications and analyses [80, 103, 105, 109]. Typically, a spectral clustering algorithm has these following steps:

1. Define a specific Laplacian matrix.
2. Find the $k$ eigenvectors corresponding to the smallest eigenvalues.

3. Embed data onto a $k$-dimensional space according to the eigenvectors.
4. Obtain cluster labels with a $k$-means algorithm.[1]

While spectral clustering is noted as an elegant way of clustering non-linearly separable data and is shown to be effective on many datasets, these "classical" spectral clustering approaches have two major shortcomings:

1. Finding eigenvectors of a large matrix is computationally costly. It takes $O(n^3)$ in general using a full eigendecomposition method, and even with fast approximating techniques like IRAM (which can run into convergence issues), much space and time are required for larger datasets[2].
2. Selecting which are "informative" eigenvectors *a priori* result in embeddings that fail to capture cluster in datasets with significant noise. Spectral clustering methods usually select the eigenvectors corresponding to the largest ($A$) or the smallest (Laplacian of $A$) eigenvalues, before the $k$-means step. This is problematic when there is significant noise or outliers in the data. A selected eigenvector may correspond to a particularly salient noise or outliers, while a discarded eigenvector may contains good cluster indicators. This prompted much work on selecting "good" eigenvectors and dealing with noise in spectral clustering [63, 105, 109]. However, note that for every additional eigenvector that is evaluated for its quality, more computational cost is incurred for both finding the eigenvector and evaluating it.

PIC is similar to spectral clustering in that it finds a low-dimensional embedding of data, and a $k$-means algorithm is used to produce the final clusters. But as the results in Section 2.6 have shown, it is *not* necessary to compute *any* specific eigenvector in order

[1] Other methods can be substituted in the place of $k$-means. For example, simply threshold one eigenvector to get two clusters, and recursively partition the clusters until $k$ clusters are obtained [91]. $k$-means is often used because the method is simple to implement, effective in practice, and quite scalable.

[2] For a sparse affinity matrix, IRAM has a space complexity of $O(e) + O(nm)$ and a time complexity of $(O(m^3) + (O(nm) + O(e)) \times O(m - k)) \times r$; $n$ is the number of nodes, $e$ is the number of edges, $m$ where $m > k$ is the *Arnoldi length* parameter, and $r$ is the number of restarts [22].

to find an eigenvector-inspired low-dimensional embedding useful for clustering. In this respect, PIC is very different from the spectral methods mentioned above.

## 3.2 Kernel *k*-means

Another graph clustering approach closely related to spectral clustering, but with substantial speed improvement is *multilevel kernel* k*-means* [31], where the general *weighted kernel k-means* is shown to be equivalent to a number of spectral clustering methods in its objective when the right weights and kernel matrix are used. Performance wise, spectral clustering methods are slow but tend to get globally better solutions, whereas kernel k-means is faster but is more easily stuck in a local solution. Multilevel kernel k-means exploits this trade-off using a multilevel approach: first an iterative coarsening heuristic is used to reduce the graph to one with $5 \cdot k$ nodes where k is the number of desired clusters. Then spectral clustering is used on this coarse graph to produce a base clustering, and then the graph and is refined iteratively (to undo the coarsening), and at each refinement iteration the clustering results of the previous iteration is used as the starting point for kernel k-means. Additional point-swapping can be used to further avoid being trapped in local minima. Compared to PIC, multilevel kernel k-means is a much more complex method, and similar to sampling [22, 38, 107] and a multilevel [97] spectral methods, eigencomputation on the sampled or "coarse" version of the data will still in general have a polynomial complexity.

## 3.3 Clustering with Matrix Powering

PIC is inspired by spectral clustering, and specifically, Normalized Cuts [91], but instead of the normalized Laplacian matrix $L = I - D^{-1}A$, it operates directly on the normalized affinity matrix $W = D^{-1}A$. Taking $W$ to a certain power (in the case of PIC, the number

of iterations) leads to a *diffusion process* (see Section 3.5 for further details) that provides useful signal for clustering. Previous work have also used powers of $W$ for clustering.

In [96] powers of $W$ is viewed as a Markovian relaxation process; and when cluster signals emerge with observation of the decay of mutual-information, clusters are extracted using the information bottleneck method [95]. In [112] a simpler matrix powering algorithm is presented; instead of $W$, it uses the unnormalized affinity matrix $A$. It is not clear how well it works due to the lack of results on real datasets. Gaussian blurring mean-shift [77, 78] is yet another clustering method that involves matrix powering; however, it is significantly different from the previous two in that (1) the affinity matrix is typically a Gaussian kernel similarity manifold, and (2) the resulting clusters lie in the original data space—typically a continuous Euclidean space (unlike network data).

The main difference between PIC and the above matrix powering methods is that the core operation of PIC is the multiplication of a sparse matrix with a vector (or a few vectors) instead of matrix-matrix multiplications. In the case of [96] and [112], even if $W$ is sparse, $W^t$ quickly becomes dense even with a small $t$, and then the time complexity for each iteration is $O(n^2)$ in terms of the number of data points—prohibitively expensive even for medium-sized datasets. In the case of [77], the Gaussian blurring manifold is typically a dense affinity matrix; and if the original data $X$ does not lie in a low-dimensional space (e.g., network data) it will also require $O(n^2)$ time for each iteration. A summary comparing these matrix powering methods is shown in Table 3.1.

| Method | Matrix | Iterate | Stop | Cluster |
|---|---|---|---|---|
| Markov relaxation [96] | $W = D^{-1}A$ | $W^{t+1} = W^t W$ | information loss | inf. bottleneck |
| Matrix powering [112] | $A$ | $A^{t+1} = A^t A$ | a small $t$ | thresholding |
| GBMS [77] | $W = D^{-1}A$ | $X^{t+1} = WX^t$ | entropy | thresholding |
| PIC [64] | $W = D^{-1}A$ | $\boldsymbol{v}^{t+1} = W\boldsymbol{v}^t$ | acceleration | k-means |

Table 3.1: A summary of clustering methods using matrix powering. *Matrix* is the definition of powered matrix, *Iterate* is the calculation done in each iteration of the method, *Stop* indicates the iteration stopping criteria, and *Cluster* is the final cluster label assignment method.

## 3.4 Dimensionality Reduction

As discussed in Section 3.3, instead of embedding the data points $X$ in its original space, PIC embeds them in a one-dimensional or a low-dimensional space. Therefore the PIC is related to *dimensionality reduction* methods. There exists many techniques for dimensionality reduction, and their results are applied to a wide variety of tasks. Here we will discuss some of the commonly used ones and how they are related to PIC.

### 3.4.1 Latent Semantic Analysis

Latent semantic analysis (LSA) [28], also called latent semantic indexing (LSI) [29, 84] in the context of information retrieval, is a method used in natural language processing to reduce the dimensionality of text data where the instances are vectors in high-dimensional space.

If our dataset is a collection of documents and each document is represented by a vector of word counts, then the dataset can be represented as a matrix $X$ where the rows correspond to documents and columns correspond to the words and $X(i, j)$ is the number of times word $j$ appears in document $i$.[3] The document-word matrix can be factorized via singular value decomposition (SVD):

$$X = U\Sigma V^T \tag{3.1}$$

The singular values in the diagonal matrix $\Sigma$ can be interpreted to correspond to the *latent topics* in the document collection, and the magnitude of these non-negative real values indicate the "importance" of these topics. The left singular vector matrix $U$ is the document-topic projection matrix and the right singular matrix $V$ is the word-topic projection matrix. We can then construct a matrix $V_k$ by selecting columns of $U$ that

[3] Often in the SVD literature the data matrix is instead $X^T$ (rows are words and columns are documents), thus the mathematical results will be slightly different.

correspond to the largest $k$ singular values in $\Sigma$, and matrix $XV_k$ is the dataset in the lower dimensional topic space.

### 3.4.2 Principal Component Analysis

Closely related to LSA is principal component analysis (PCA) [85]. When applied to text data, PCA finds eigenvectors (principal components) and eigenvalues corresponding to the document-document covariate matrix $XX^T$ or word-word covariate matrix $X^TX$. Eigendecomposition $X^TX = E\Delta E^T$ gives us eigenvectors $E$ and eigenvalues on the diagonal of $\Delta$, and we can construct a projection matrix $E_k$ by selecting columns of $E$ corresponding to the largest $k$ eigenvalues, then the dimension-reduced data is $XE_k$.

The relationship between SVD and PCA is made evident by the following equation:

$$X^TX = (U\Sigma V^T)^T(U\Sigma V^T) = (V\Sigma^T U^T)(U\Sigma V^T) = V\Sigma^T\Sigma V^T \tag{3.2}$$

Since $X^TX$ is an invertible, real, symmetric matrix and $\Sigma^T\Sigma$ is diagonal, $V$ contains eigenvectors of $X^TX$. This implies that, if for LSA and PCA we select "interesting" dimensions based on the magnitude of the values in $\Sigma$ and $\Delta$, respectively, then $XV_k$ and $XE_k$ would yield the same dimension-reduced data.

Note that when selecting dimensions based on $\Delta$ we get dimensions corresponding to the greatest variance in data (thus dimensions with the greatest variances are preserved in the projection). As we will see later, this is not always desirable.

### 3.4.3 Random Projection

Random projection (RP) is based on the Johnson-Lindenstrauss lemma [49]: if points in a vector space are projected onto a random subspace of an appropriate dimension (high enough to capture interesting structure and low enough to filter out noise, outliers, and uninteresting features) , then the distances between the points are approximately

preserved. The process can be as simple as multiplying $X$ by a random $d \times k$ matrix $R$ with unit-length rows. The resulting $XR$ is, strictly speaking, not a proper projection because the rows of $R$ are generally not orthogonal. However, it has been shown that in a high-dimensional space, there is a sufficient number of almost-orthogonal directions so that $RR^T$ is approximately an identity matrix [46], making $XR$ a reasonable pair-wise distance preserving low-dimensional projection of $X$.

It has been suggested that random projection may be used in a two-step process to speed up LSA [84]. First apply random projection to $X$ to obtain $\hat{X} = XR$, and then apply LSA to $\hat{X}$ to obtain the final projection $X\hat{V}_k$. Since $\hat{X}$ is much smaller than $X$, SVD on $\hat{X}$ is much faster than on $X$. The second step may not be necessary, however, if we are simply looking to reduce the dimensionality of the data (as opposed to finding "latent topics") in vectorized image and text datasets [16]. We will see later in Section 3.5 how this random mapping is related to PIC's initial vector $\boldsymbol{v}^0$.

**Random Projection and Clustering**

Two noted characteristics of random projections are: (1) some high-dimensional distributions "look more Gaussian" when projected onto a low-dimensional space, and (2) the projection can change the shape of highly eccentric clusters to be "more spherical" [26]. These observations make random projections particularly suitable for clustering by fitting the data to a Gaussian mixture model (GMM). However, experiments have shown that clustering results from these random projection with Gaussian mixture models (RP-GMM) can be quite unstable, perhaps due to both the random directions selected in RP and random initial distributions for the expectation maximization (EM) procedure in GMM. Therefore it is recommended that a stabilization technique is used, such as using an ensemble of RP-GMMs and unifying the results with agglomerative clustering [35].

**Random Mapping**

Presented independently in [52] but also motivated by the near-orthogonality of random vectors in high-dimensional space shown in [46], like random projection, random mapping first projects $X$ onto a lower dimensional space using a random matrix $R$ where the rows are unit-length random vectors, and then $XR$ is fed into a self-organizing map (SOM) [54] to produce a (usually two-dimensional) discretized, intuitive representation of the data for exploratory analysis and visualization [54].

### 3.4.4 Locality Sensitive Hash

Locality sensitive hash (LSH) [21] is a hashing method closely related to RP that is often used to find the nearest neighbors of a data point in vector space. A basic LSH algorithm is as follows:

1. Obtain $\hat{X} = XR$ as in random projection
2. Create a bit matrix $H$ where $H_{i,j} = \begin{cases} 1 & \text{if } \hat{X}_{i,j} > 0 \\ 0 & \text{else} \end{cases}$

Instead of a projection onto a low-dimensional space, LSH projects (hashes) each data point onto a bit vector. In the context of LSH, each column of $R$ can be viewed as a hyperplane, and each component of the resulting bit vectors tells us on which side of the hyperplane the data points lie. It is not difficult to see that the Hamming distance between two bit vectors is closely related to the cosine of the angle between the two corresponding vectors. Specifically:

$$\text{Prob}(H_{i,k} = H_{j,k}) = 1 - \frac{\theta(X_i, X_j)}{\pi} \tag{3.3}$$

Thus the Hamming distance between signatures can be used as an approximation of the cosine distance between two data points. Because the signatures are usually a much

more compact representation than the original vectors, LSH can be useful for efficiently detecting and retrieving similar items in high-dimensional vector space. Analogous to random indexing for random projection, there are also online versions of LSH [99, 100].

## 3.5 Diffusion Maps

Unlike LSA, PCA, RP, and RI, which are linear dimensionality reduction methods, diffusion maps (DMs) [58] are a method for non-linear dimensionality reduction. Instead of operating directly on Euclidean geometry of the vector space of the input data, it is based on the *connectivity* of the data points. The connectivity between data points is defined via a similarity function (a diffusion kernel) $s(X_i, X_j)$ with the following properties:

1. $s(X_i, X_j) = s(X_j, X_i)$ (i.e., symmetric)
2. $s(X_i, X_j) \geq 0$ (i.e., non-negative)

If the dataset $X$ are points in vector space, $s$ may be a Gaussian kernel:

$$s(X_i, X_j) = e^{-\frac{\left\|X_i - X_j\right\|^2}{2\sigma^2}} \tag{3.4}$$

which defines a neighbor around each data point in $X$ parameterized by the $\sigma$, the bandwidth. If the dataset is a simple undirected graph or a network, where each data point is a node, then $s$ may be defined as follows:

$$sX_i, X_j = \begin{cases} 1 & \text{if an edge exists between } X_i \text{ and } X_j \\ 0 & \text{else} \end{cases} \tag{3.5}$$

In the context of DMs, it is often more convenient to see the dataset as a graph or a network and talk about the relationship between data points in terms of their *connectivity* or *similarity*.

Given a dataset $X$, we define an affinity matrix $A$ where $A_{ij} = s(X_i, X_j)$ for some similarity function $s$ and a diffusion matrix $W = D^{-1}A$. So $W$ is then a row-normalized stochastic matrix, defining transition probabilities on a Markov chain on $X$; in other words, $W$ defines probabilities for discrete-time random walk on data points in $X$ weighted according to $A$.

### 3.5.1 Diffusion Process

The diffusion matrix $W$ defines the transition probabilities for a single time step, and equivalently, Markov chains of length one. If we multiply $W$ by itself, i.e., $W^2$, then $W^2_{i,j}$ would give us the probability for transitioning from $i$ to $j$ in two time steps. Taking that a step further, $W^t_{i,j}$ gives us the probability for randomly "walking" from $i$ to $j$ in $t$ time steps according to similarity function $s$. It is important to note that while $t$ is the time parameter in this *diffusion process*, it should be also viewed as a *scale* parameter—the larger it is, the longer the Markov chain, and the more $W^t$ gives us a coarse-grained look at the connectivity between data points in $X$.

### 3.5.2 Diffusion Distance and Diffusion Space

If the transition probabilities in $W^t$ give us the underlying geometry of the data, it is useful to define a distance metric $d_t$ in terms of $W$, such that:

$$d_t(X_i, X_j)^2 = \sum_k \left| W^t_{ik} - W^t_{jk} \right|^2 \tag{3.6}$$

This definition is rather intuitive—according to this metric, in order for $i$ and $j$ to be "close" to each other, the length-$t$ path probability from $i$ to an arbitrary data point $k$, and the length-$t$ path probability from $j$ to $k$, should be roughly the same. Putting it in terms of Markov random walks, if the nodes that are unlikely reached by walks from $i$ are also unlikely reached from $j$, and the nodes that are likely reached by walks from

$i$ are also likely reached by $j$, then the distance between $i$ and $j$ is small according to $d_t$. If $i$ and $j$ are the same node then $d_t = 0$. We call this distance metric the *diffusion distance* [23,58].

An important and interesting characteristic of the diffusion distance is that it is very much related to the Euclidean distance. In fact, if we view the rows of $W^t$ as vectors and the columns of $W^t$ as coordinates in vectors space, then $d_t$ is the same as the Euclidean distance between vectors in the space spanned by the rows of $W^t$. We call this space the *diffusion space*, defined by the data points $X$, the similarity function $s$, and the scale parameter $t$.

### 3.5.3 Diffusion Maps and Dimensionality Reduction

We can now define a mapping based on the previous sections: a *diffusion map* is the mapping of a data point from $X$ to a coordinate in the diffusion space associated with $W^t$. This mapping is useful because we are now able to analyze the data with tools from vector space and Euclidean geometry; for example, we can apply a k-means algorithm to the mapped data for cluster analysis. However, from a practical and computational point of view, the full diffusion space has as many dimensions as the number of data points and therefore it is difficult to visualize, explain, and it is also computationally expensive to store and process.

Given the high dimensionality of the diffusion space in most practical applications, diffusion maps is most useful when the dimensionality of the target space is *reduced*. It turns out, much like PCA, the dimensions that best preserve the pair-wise distances between points in the diffusion space are related to the eigenvectors and eigenvalues of

$W$ [23, 27]. Specifically, the family of diffusion maps $\{\Psi_t\}$ is given by:

$$\Psi_t(x_i) := \begin{pmatrix} \lambda_1^t e_1(i) \\ \lambda_2^t e_2(i) \\ \vdots \\ \lambda_d^t e_d(i) \end{pmatrix} \tag{3.7}$$

where $d$ is the desired number of dimensions and $\lambda_0, \lambda_1, \lambda_2, \ldots$ are the eigenvalues of $W$, in order from the largest to the smallest, and $e_0, e_1, e_2, \ldots$ are the corresponding eigenvectors. Note that $\lambda_0$ and $e_0$ are omitted from the mapping because $e_0$ is a constant vector and therefore uninteresting in a dimensionality reduction setting.

The definition for $\Psi_t$ also shows us how the scale parameter $t$ affects the mapping—it exponentially scales the coordinates in the dimension represented by an eigenvector according to the corresponding eigenvalue. An interesting and useful corollary is that, as $t$ increases, $d$, the number of dimensions required to accurately represent the data in a low-dimensional space, decreases, unless the eigenvalues are nearly the same. For detailed analyses, proofs, and additional information on diffusion maps see [23, 27, 58].

## 3.6 Diffusion, Random Projection, and PIC

PIC is related to spectral clustering, and in particular, to Normalized Cuts. Both methods project the data from the original space onto a lower-dimensional space based on the eigenvectors of $D^{-1}A$, and then uses a k-means method to output cluster labels. Here we will see that PIC is also closely related to diffusion maps and random projection. PIC, like spectral clustering methods, first embeds the data points from a pair-wise similarity manifold onto a low-dimensional vector space before assigning clusters either by thresholding or by a clustering method like k-means. We will refer to the PIC low-dimensional embedding as the *power iteration embedding* (PIE).

Since PIEs are typically constructed with one or a few random vectors, it is related to random projection, which also make use of random vectors in high-dimensional spaces. And like PIE, diffusion maps also allow us to represent non-linear, high-dimensional data in linear, low-dimensional spaces. Recall that in power iteration:

$$\boldsymbol{v}^t = W\boldsymbol{v}^{t-1} = W^2\boldsymbol{v}^{t-2} = ... = W^t\boldsymbol{v}^0 \tag{3.8}$$

where $W = D^{-1}S$ and $\boldsymbol{v}^0$ is a random vector and $\boldsymbol{v}^t$ is the vector at the t-th power iteration. As mentioned in Section 2, to avoid *cluster collisions* in one dimension, we can run power iteration d times using d independently generated random vectors and create a d-dimensional PIE on which to do clustering. If we make them all stop at the same t, we can express it as:

$$E = W^t R \tag{3.9}$$

where R is a $n \times d$ matrix that is the concatenation of randomly generated vectors and E is PIE, the low-dimensional embedding in PIC. Note that here $W$ is the same diffusion matrix $W$ in Section 3.5, and recall that the t of $W^t$ is a both a time and scaling parameter; it is a time parameter in the diffusion process, and it is a scaling parameter in revealing the underlying geometric structure of the data at different scales. If we restrict the generation of R to be zero-mean and unit-variance, then we have the following result:

> *The power iteration embedding* E *is a random projection of data points of* X *in the diffusion space defined by the similarity function* s *and the scale parameter* t.

To illustrate this connection, we can imagine a procedure for finding clusters in a dataset X, shown as Algorithm 3. This is a reasonable way to find clusters in data in a non-linear manifold. Many clustering algorithms, such as k-means, do not work well (or at all) in non-linear spaces or high-dimensional spaces (due to the curse of dimensional-

ity [15]), but diffusion maps allow us to work with non-linear data in Euclidean space and random projection is an efficient alternative to PCA or SVD for dimensionality reduction. However, if the number of data points $n$ is large, $W^t$ is prohibitively expensive to calculate and store ($O(n^2)$).

**Algorithm 3** Diffusion+Random Projection+k-means

1: **procedure** DIFFRPCLUSTER($X, k, s, t$)
2: Construct the diffusion matrix $W$ from dataset $X$ and similarity $s$
3: Raise $W$ to the power of scale $t$ to embed data in the diffusion space $W^t$
4: Determine $d$, the number of dimension desired
5: Reduce the dimensionality of $W^t$ using random projection to get projection $E$
6: Run k-means on $E$ to get clusters $C_1, C_2, ..., C_k$.
7: **return** $C_1, C_2, ..., C_k$
8: **end procedure**

On the other hand, PIC will produce the exactly same result as the above method with storage and running time linear to the size (number of edges or non-zero features) of the data. In addition, in PIC $t$ can be determined efficiently on-the-fly, avoiding parameter sweeps.

# Chapter 4

# MultiRankWalk

## 4.1 Introduction

Traditional machine learning or *supervised learning* methods for classification require large amounts of labeled training instances, which are often difficult to obtain. In order to reduce the effort required to label the training data, two general solution have been proposed: *semi-supervised learning* methods [113] and *active learning* methods (e.g., [39, 115]). Semi-supervised learning (SSL) methods learn from both labeled *and* unlabeled instances, reducing the amount of labeled instances needed to achieve the same level of classification accuracy. Active learning methods reduce the number of labels required for learning by intelligently choosing which instances to ask to be labeled next.

Many semi-supervised learning methods fall under the category of *graph-based semi-supervised learning* [13, 45, 71, 92, 93, 111, 114], where instances are nodes in a graph and similarities between the instances define weighted edges. This representation is useful and powerful; most datasets can be represented as a graph through a pair-wise similarity manifold and many existing graph algorithms and theories can be readily applied. Table 4.1 lists symbols we will use in the context of semi-supervised learning.

| Symbol | Definition |
|---|---|
| $\mathcal{X}$ | The dataset where $\mathcal{X} = \mathcal{X}^{\mathrm{L}} \cup \mathcal{X}^{\mathrm{U}}$ and $\mathcal{X}^{\mathrm{L}} \cap \mathcal{X}^{\mathrm{U}} = \oslash$ |
| $\mathcal{X}^{\mathrm{L}}$ | Labeled "seed" instances in $\mathcal{X}$ |
| $\mathcal{X}^{\mathrm{U}}$ | Unlabeled instances in the dataset $\mathcal{X}$ |
| $y^{\mathrm{L}}$ | Class labels for $\mathcal{X}^{\mathrm{L}}$ |
| $y^{\mathrm{U}}$ | Class labels for $\mathcal{X}^{\mathrm{U}}$ |

Table 4.1: Symbol definitions for semi-supervised learning.

As the number of proposed graph-based SSL methods increases, many questions remain unanswered. How do these methods relate to each other? Which methods do better, under what condition, and for what type of data? What method should be use as a strong baseline when working on a particular type of data? Here we aim to address some of these issues, with a special focus on when there are very few labels.

First, we describe a semi-supervised learning method based on random graph walk and relate it to methods that fall under a class of graph walk-based algorithms, such as [42, 45, 111]. The core computation of these methods usually involves finding the dominant eigenvector of some form of affinity matrix or transition matrix of the graph. The proposed method is probably simplest of them all, yet it is also intuitive, scalable, and extremely effective on a number of network datasets, leading us to highly recommend the proposed graph walk-based method as a strong baseline for future research in semi-supervised learning.

Second, for reducing the cost of obtaining instance labels, one issue has not been considered: that in many practical settings, *some instances are easier to label* than others. For example, in classifying websites, a better-known website is very likely easier for a domain expert to label, since the expert would be more likely to be familiar with it, and since the website would be less likely to be difficult-to-evaluate because of having content that is limited (or simply incoherent). In selecting seeds (labeled instances), we evaluate using highly authoritative instances. In addition to being arguably easier to label, these authoritative instances are arguably more likely to spread their influence (and

their labels) to their neighbors in a network (assuming homophily), therefore making them better seeds in a semi-supervised learning setting.

## 4.2 Semi-Supervised Learning with Random Walks

MultiRankWalk (MRW) is an iterative, label-propagation semi-supervised learning method based on random graph walk [65]. Its core algorithm is directly related to personalized PageRank [44, 82] and random walk with restart (RWR) [98]. Given a graph representation $A$ with $n$ nodes, a vector of random walk probability distribution over the nodes $\boldsymbol{\nu}$ with $|\boldsymbol{\nu}| = n$ satisfies the following equation:

$$\boldsymbol{\nu} = (1 - d)\mathbf{r} + dP\boldsymbol{\nu} \tag{4.1}$$

where $P$ is the column-stochastic transition matrix of graph $A$, $\mathbf{r}$ is a normalized restart vector where $|\mathbf{r}| = n$ and $\|\mathbf{r}\|_1 = 1$, and $d$ is the damping factor. The vector $\boldsymbol{\nu}$ can be solved for efficiently by iteratively substituting $\boldsymbol{\nu}^t$ with $\boldsymbol{\nu}^{t-1}$ until $\boldsymbol{\nu}^t$ converges. The vector $\boldsymbol{\nu}$ can be interpreted as the probability distribution of a random walk on $A$, with teleportation probability $1 - d$ at every step to a random node with distribution $\mathbf{r}$. We define the ranking vector $\boldsymbol{\nu}$ as a function of $A$, $\mathbf{r}$, and $d$:

$$\boldsymbol{\nu} = RandomWalk(A, \mathbf{r}, d) \tag{4.2}$$

The main difference between many random walk algorithms lies in the use and interpretation of $\mathbf{r}$. For PageRank [82], where $A$ is the connectivity matrix of a network of hyperlinked web pages, $\mathbf{r}$ is simply a uniform vector; with probability $1 - d$ a web surfer tires of following the links he sees and jump to a random page. For personalized PageRank [44], each web surfer, instead of jumping to a random page, jumps to a page according to his or her unique preference—encoded as a multinomial distribution vector

$\mathbf{r}$. For random walk with restart (RWR) [98], $\mathbf{r}$ is an all-zero vector except at position $i$, where $\mathbf{r}_i = 1$; $i$ is the *starting node*; at every time step the random walker either follows an edge with probability $d$ or jumps back to $i$ (*restarts*) with probability $1 - d$.

In this work the nodes of $A$ are instances and edges represent similarity or affinity between the instances. Labeled training instances of each class are described by the vector $\mathbf{r}$, the *seed vector*, where each non-zero element corresponds to a labeled training node. The random walk describes classification as a process of finding unlabeled instances similar to the labeled instances based on random walk probability. For each class $c$, at every time step the search process may follow an edge with probability $d$ or it may decide to restart the process at an instance labeled $c$ with probability $1 - d$. This process is repeated for each class and an unlabeled instance is labeled according to the class whose process most often visited it.

**Algorithm 4** The MultiRankWalk algorithm

**procedure** MRW($A, Y^L, \alpha$)
  $R_{ci} \leftarrow 1$ if $Y^L_i = c$, else $R_{ci} \leftarrow 0$
  Normalize columns of $R$ to sum to 1
  $V^0 \leftarrow R$ and $t \leftarrow 0$
  **repeat**
    $V^{t+1} \leftarrow (1-\alpha)AD^{-1}V^t + \alpha R$ and $t \leftarrow t+1$
  **until** $V^t$ has converged
  $Y^U_i \leftarrow argmax_c(V_{ci})$
  **return** $Y^U$
**end procedure**

Shown as Algorithm 4, we call it *MultiRankWalk* because it does classification by creating multiple rankings using random walks from seed instances. This method was developed independently, but it is similar to several previously described methods or one of their variations [40, 42, 111] in the way labels are propagated from the "seeds" to the rest of the instances. Details of the difference between these methods is discussed in Section 5.

## 4.3 Seed Selection

Semi-supervised learning methods require labeled training instances as *seeds*, and we propose using more "authoritative" instances (e.g., blogs that are frequently referred to and cited by other blogs). There are two reasons to prefer highly authoritative instances as seeds:

1. Popular or authoritative instances are easier to obtain labels for because a) domain experts are more likely to recognize them and label quickly, and b) they are more likely to have already labels available.
2. Popular or authoritative instances will likely have many incoming links and sometimes outgoing links as well. Having many incoming and outgoing links may help to propagate the labels faster and more reliably when using a graph-based semi-supervised learning method.

Based on these assumptions, we propose a general seed labeling strategy to test our hypothesis: RANKED-AT-LEAST-M-PER-CLASS. This strategy takes as input an ordered list of instances according to a ranking preference, the highest rank instance first. Given an integer $m$, we start at the top and label each instance as a seed instance going down until we have at least $m$ seeds per class labeled as seed instances. It simulates a domain expert labeling an ordered list of items and labeling instances one by one until he or she runs out of time and stops.

For all experiments in the next section we vary $m$ and test these different seed ranking preferences:

- Random is a baseline ranking where the order is uniformly random.
- LinkCount ranks the instances based on the number of connecting edges; instances with higher number of edges are ranked higher.
- PageRank ranks the instances based on the PageRank citation ranking algorithm [82]; nodes with higher PageRank scores are ranked higher.

Note that all three of these ranking preferences can be efficiently calculated for most network datasets.

## 4.4 Experiments

### 4.4.1 Datasets

To assess the effectiveness of our method, we test it on five network datasets. The first three datasets are from the political blog domain: UMBCBlog, AGBlog, and MSPBlog. The other two are from the scientific paper citation domain: Cora and CiteSeer. All of these datasets contain explicit links between the instances in the form of hyperlinks or citations. In constructing the graph from these datasets we take the simplest approach—each dataset is a graph containing undirected unweighted edges:

- UMBCBlog is constructed as in [51]: first we find a set of overlapping blogs between the ICWSM 2007 BuzzMetrics [3] dataset and the labeled dataset in [5], then a graph is formed using links found in the BuzzMetrics dataset posts, and lastly we take the largest connected component of the graph as the affinity matrix $A$, with nodes labeled *liberal* or *conservative*.
- AGBlog is the largest connected component of the political blog network found in [5], again with nodes labeled either *liberal* or *conservative*. Although the nodes and labels in UMBCBlog are a subset of those in AGBlog, the edges in these two datasets are very different. In UMBCBlog, the links are gathered in May 2006 from the content of the blog posts; in AGBlog, the links are from two months before the 2004 presidential election and are extracted from the sidebars of the blogs [5]. The links from UMBCBlog can be considered to indicate the bloggers' interests at the time of the post, and links from AGBlog can be considered to indicate the bloggers' long-term interests and recommendations.

- MSPBlog is provided by the researchers at Microsoft Live Labs and is constructed separately from the above two datasets. From a large collection of automatically crawled news and blog sites, a fraction of the politically themed blogs are manually labeled either *liberal* or *conservative*.
- Cora is a scientific paper citation dataset that contains papers from 7 related but distinct scientific fields. An edge exists in the graph between node a and node b if paper a cites paper b or vice versa. The details of its construction can be found in [69].
- CiteSeer is another scientific paper citation dataset from [69] and contains papers from 6 related scientific fields. Summary statistics of these five datasets can be found in Table 4.2 and class distribution in Tables 4.3 and 4.4.

| | Nodes | Edges | Density |
|---|---|---|---|
| UMBCBlog | 404 | 2725 | 0.01670 |
| AGBlog | 1222 | 19021 | 0.01274 |
| MSPBlog | 1031 | 9316 | 0.00876 |
| Cora | 2485 | 5209 | 0.00084 |
| CiteSeer | 2110 | 3757 | 0.00084 |

Table 4.2: Node count, edge count, and density of the network datasets. Density is the ratio of number of edges to the number of nodes squared.

| | UMBCBlog | AGBlog | MSPBlog |
|---|---|---|---|
| Liberal | 198 | 586 | 375 |
| Conservative | 206 | 636 | 656 |

Table 4.3: Class distribution for the political blog datasets.

### 4.4.2 Setup

We compare MRW against the harmonic functions method of Zhu et al. [71, 114] on five network datasets and the effect of different seed preferences on these learning algorithms. We vary the number of labeled instances by changing the seeding parameter

| Cora | | CiteSeer | |
|---|---|---|---|
| Neural Networks | 726 | HCI | 304 |
| Case Based | 285 | IR | 532 |
| Reinforcement Learning | 214 | Agents | 463 |
| Probabilistic Methods | 379 | AI | 115 |
| Genetic Algorithms | 406 | ML | 308 |
| Rule Learning | 131 | DB | 388 |
| Theory | 344 | | |

Table 4.4: Class distribution for the citation datasets.

$m$ mentioned in Section 4.3 with $m = 1, 2, 5, 10, 20$, and 40. The reported numbers for random seeding are averaged over 20 runs.

The harmonic functions method (HF) is a simple label propagation algorithm that estimates class-membership probabilities by assuming the existence of homophily. It was proposed in [114] to be used with Gaussian fields as a semi-supervised learning method for general data. HF is "essentially identical" to the wvRN method of Macskassy and Provost [71] except with a principled semantics and exact inference (for details see Section 5), and is one of the best methods for classification on a number of benchmark network datasets in [71]. Note that in our experiments the implementation of HF is the one described in [71] for wvRN due to its efficiency on sparse networks, but we will refer to it as HF since it came first and is more widely cited. Many other recent SSL methods use a similar approach [13, 90, 93]; and they all generally produce very similar results. We compare MRW's performance with HF (and transitively, with wvRN as well) on the five network datasets using different seed preferences.

### 4.4.3 Results

#### Comparing SSL Methods and Seeding Preferences

We use both classification accuracy and macro-averaged F1 according to the ground truth labels as performance measures. For definitions of these measures see Appendix A.1.

Here for discussion we will focus on the F1 score; for accuracy scores and additional F1 score charts see Appendix B.1. The F1 score is usually preferred over accuracy as an evaluation measure when the class label distribution is unbalanced, which is true of most of these datasets.

In the results shown F1 score is associated a particular learning algorithm, seeding preference, seed parameter $m$, and dataset. Figure 4.1 shows scatter plots of HF F1 scores versus MRW F1 scores. A point lies above the center diagonal line when MRW F1 score is higher than that of HF, and vice versa. The two scatter plots are identical except the points are marked differently; the left according to the seeding preference and the right according to the number of training labels as determined by $m$. Figure 4.2 shows bar charts of the differences between MRW F1 scores and HF F1 scores on different datasets and with different seeding preference and parameter $m$. A positive value means the MRW F1 score is higher than that of HF, and vice versa.

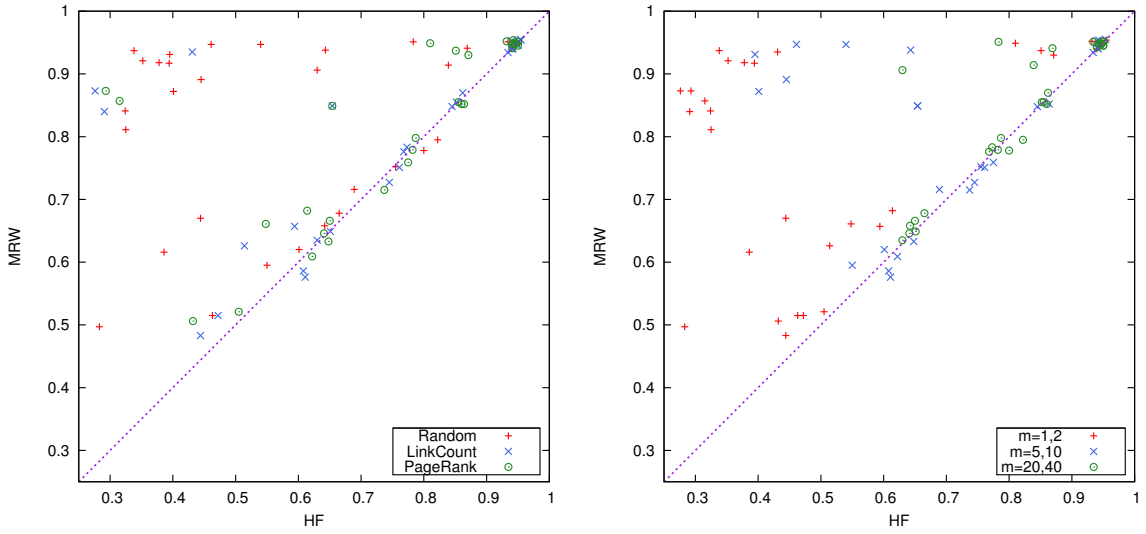



Figure 4.1: Scatter plots of HF F1 score versus MRW F1 score. The left plot marks different seeding preferences and the right plot marks varying fraction of training labels determined by $m$.

We make some observations from these figures:

1. MRW is able to achieve high classification accuracy with very few labeled instances.

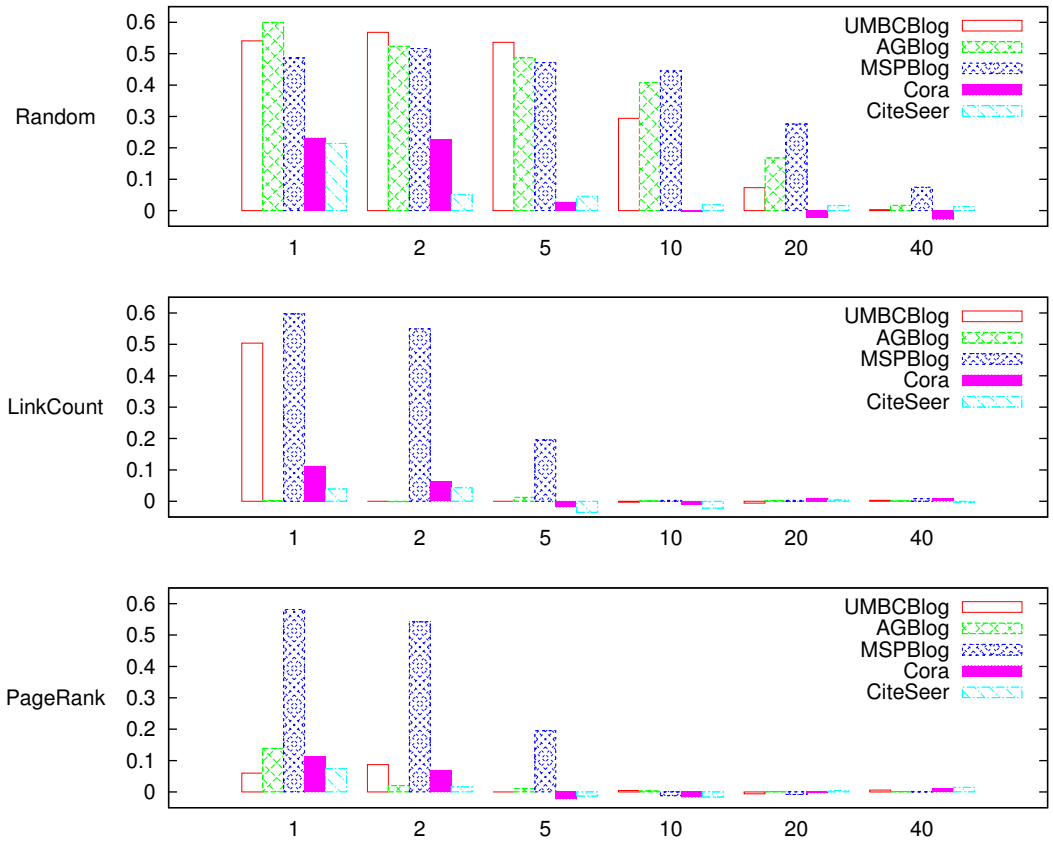



Figure 4.2: Bar charts of the differences between MRW F1 scores and HF F1 scores. The x-axis is the seeding parameter $m$ that determines the number of training labels, and the y-axis is MRW F1 score minus the HF F1 score. The top, middle, and bottom charts are results using Random, LinkCount, and PageRank seeding preferences, respectively.

2. On most datasets and seed preferences MRW outperforms HF by a large margin when the amount of training data is very small. The only exception to this is on the CiteSeer dataset when HF is paired with LinkCount or PageRank seeding. On all datasets MRW and HF F1 scores converge when the training data size reaches above 30% of the test data size.

3. The performance difference between MRW and HF is the greatest when seeds are chosen randomly. This strongly suggests that MRW is more robust to varying quality of the labeled data.

4. With HF, preferring more authoritative seeds dramatically outperforms random seeds, especially when the number of labeled instances is small; on the blog

datasets preferring more authoritative seeds reduces the amount of labeled instances required to reach the same level of performance by a factor of 40 to 50. Out of the two authority-based preferences, PageRank seems to be slightly better and more consistent in yielding quality seeds.

5. With MRW, the difference between random seeds and the authoritative seeds are not as dramatic, one reason being that on the political blog datasets the F1 is already very high with random seeding. However, a statistically significant difference is still observed on AGBlog, MSPBlog, and Cora datasets when the number of labeled instances is very small.[1]
6. When comparing LinkCount and PageRank, PageRank is a better and more stable seed preference, and the performance of different seed preferences converge when the amount of training data is large enough.

**Versus Relational Learning and Spectral Clustering** To show how much classification power can be gained from link structure alone, we compare the results of our algorithm against some supervised relational learning methods, shown in Figure 4.3. The numbers shown in the charts are accuracy scores. The algorithms are labeled on the figures as follows:

- MRW is MultiRankWalk method using PageRank seeds.
- Kou is the best result reported in [55].
- Kou-Rerun is the result from our re-run of experiments described in [55].
- Kou-Rerun-C is our re-run of [55] using only the largest connected component of the dataset.

[1]For comparing significant difference between HF and MRW when using CountLink and PageRank seed preferences, a one-tail paired McNemar?s test on the classification result of individual instances is used with $p < 0.001$ reported as significant. For comparing significant difference between HF and MRW when using Random seed preference, the 20 accuracy scores from the 20 random trials are used in an one-tail Mann-Whitney U test with $p < 0.001$ reported as significant. For comparison between the random seeding and authority-based seed preferences, the classification result of individual instances is used in an one-tail Mann-Whitney U test with $p < 0.05$.

- Lu is the best result reported in [69].
- Content-Kou is the content-only baseline result reported in [55].
- Content-Lu is the content-only baseline result reported in [69].

MRW shows outstanding performance considering the simplicity of the algorithm, the small number of labeled instances required, and the fact that it uses only the link structure. Based on these results we recommend label propagation algorithms such as MRW as a strong baseline for semi-supervised learning or supervised relational learning for network data.

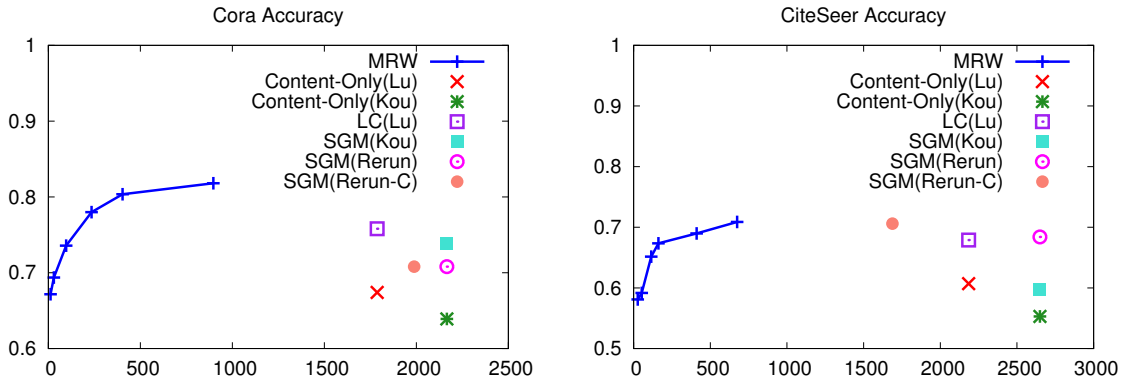



Figure 4.3: Citation datasets results compared to supervised relational learning methods. The x-axis indicates number of labeled instances and y-axis indicates labeling accuracy.

### Damping Factor

The effect of the damping factor d on the proposed learning method is shown in Figure 4.4. Generally, a higher damping factor result in slightly better classification performance. This suggests that it is better for the algorithm to propagate the labels further by not "damping" the walk too much, especially when the number of labeled instances is small.

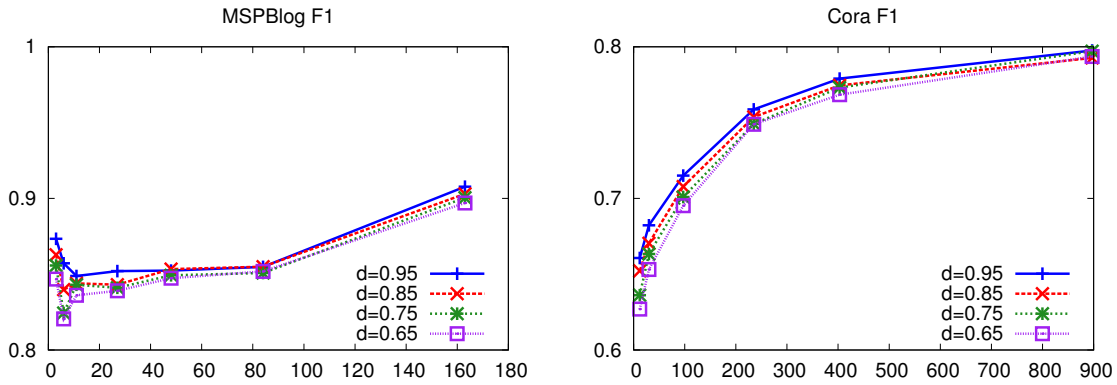



Figure 4.4: Results on three datasets varying the damping factor. The x-axis indicates number of labeled instances and y-axis indicates labeling macro-averaged F1 score.

## 4.5 Scalability

The best seed preference algorithm is based on PageRank, so the run time is linear in the number of edges in the graph and converges fairly quickly even when applied to large graphs [82]. The proposed algorithm is based on random graph walk with restart, and the run time is also linear to the number of edges in the graph; the core algorithm itself has been well-studied and several performance-enhancing methods have been proposed to minimized the amount of storage and time required such as the one found in [98].

## 4.6 Comparison with Clustering Methods

Graph-based SSL methods such as HF and MRW and graph-based clustering methods such as NCut and PIC are all methods that reply on a strong homophily assumption and are all related to random walks on graphs. Here we compare the performance of these methods, plus a few hybrid ones, on 10 benchmark social network datasets. The description of the datasets can be found in Sections 2.6, 2.7, 4.4.1. We compare 11 methods; 4 clustering methods and 7 learning methods: NCut is Normalized Cut [91]; NJW is the spectral clustering method proposed by Ng, Jordan, and Weiss [80]; PIC is power iteration clustering (Chapter 2); PIC4 is multi-dimensional PIC (Section 2.7) with

4 dimensions; HF is the harmonic fields method; MRW is MultiRankWalk; 1NN is the 1-nearest neighbor classifier applied directly to the graph data; PIE4+1NN is 1NN on a 4-dimensional power iteration embedding; SVM is the support vector machine classifier [101] applied directly to the graph data; PIE4+SVM is SVM on a 4-dimensional power iteration embedding; and PIE2k+SVM is SVM on a 2k-dimensional power iteration embedding, where k is the number of clusters. For the SSL methods we vary the amount of training/seed instances at 1%, 2%, 5% and 10%, and the rest of the dataset are test instances. SVM uses a one-versus-all scheme [72] for multi-class classification. For each method, dataset, and a training instance percentage, we run 50 trials and report the average. Figure B.7 and B.8 show the macro-average F1 scores.

As shown before according to the experiments in Section 2.6 and 2.7, PIC does well as, and sometime better, than the spectral clustering methods (NCut and NJW) on datasets with a smaller number of classes, and on datasets with a larger number of classes (Football and MGEmail) PIC4 is competitive with spectral clustering where PIC runs into "cluster collision" problems. Also as expected, according to experiments in Section 4.4, MRW outperforms HF, and it does so with a large margin when the number of seed instances is small. Besides these observations that confirm previous results, we note the following new observations:

- MRW, even with just 1% seed, is usually competitive with, and sometimes better, than clustering methods; the same cannot be said of HF.
- While 1NN does not work well when applied directly to the graph data, it is much better on the PIC embedding (PIE4+1NN); it shows that the PIC embedding is not only useful for clustering, but also for SSL. For example, on Cora PIE4+1NN has an F1 score about 0.6 even though PIC is at about 0.3.
- As expected, SVM turns out to be a very inaccurate classifier on many of the datasets when applied directly to the graph data. However, we note that although PIE4+SVM works well (better than SVM and competitive with MRW) on datasets

with a small number of classes, it does not seem to work when the number of classes gets larger (e.g., Football and MGEmail).

- When we increase the dimensionality of PIE from 4 to 2k, SVM is able to gain some accuracy, although still not as accurate as PIE4+1NN. This suggests that at PIE4, the classes are separated enough so 1NN is able to perform almost as well as MRW, but they are not separated *linearly*, a requirement of linear SVMs. When the dimensionality is increased the classes are more likely to be linearly separable, therefore SVM is more likely to find a separating hyperplane.

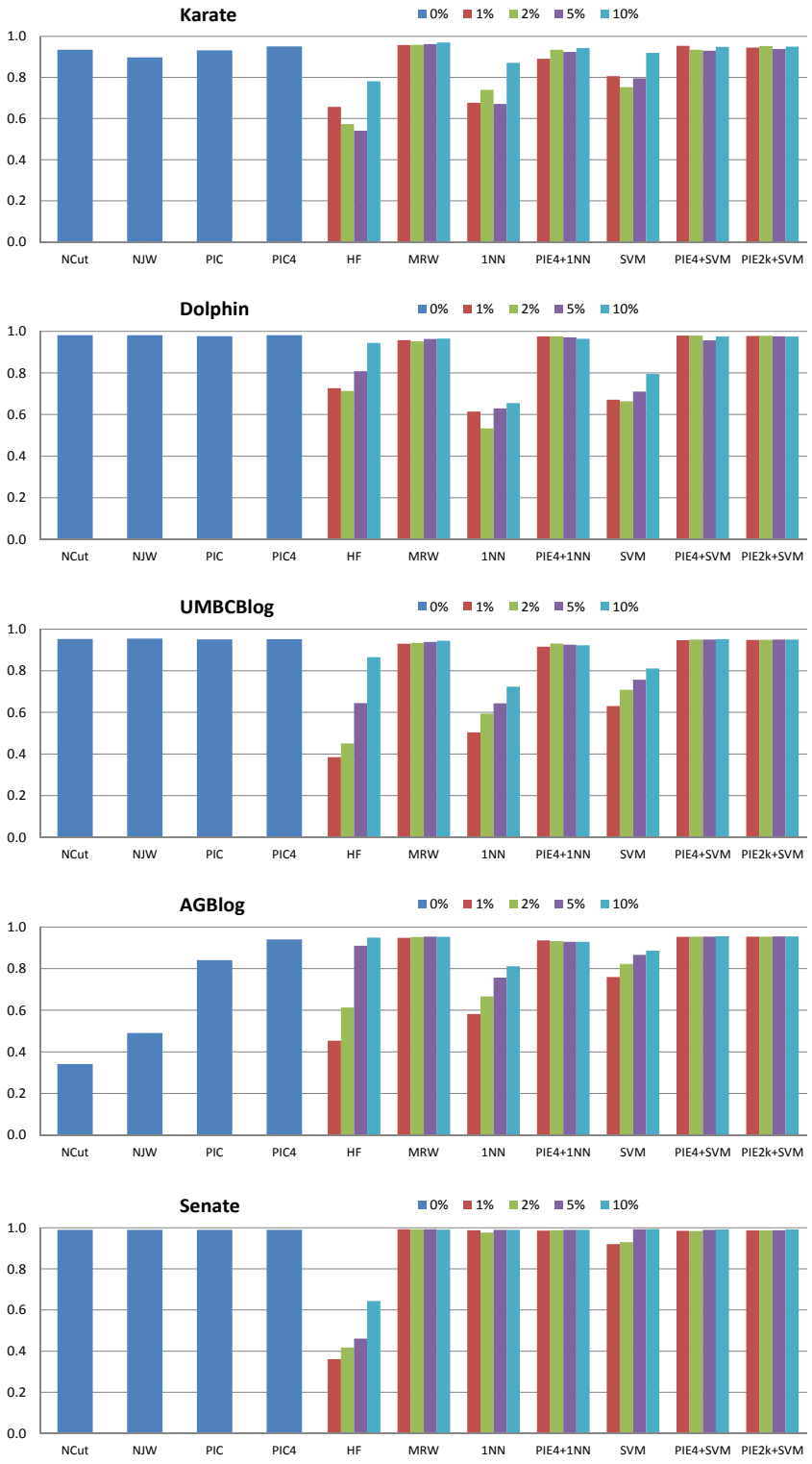



Figure 4.5: F1 comparison between SSL methods and clustering methods on network data. Colors correspond to the percentage of data used as training/seed instances; 0% corresponds to a clustering problem.

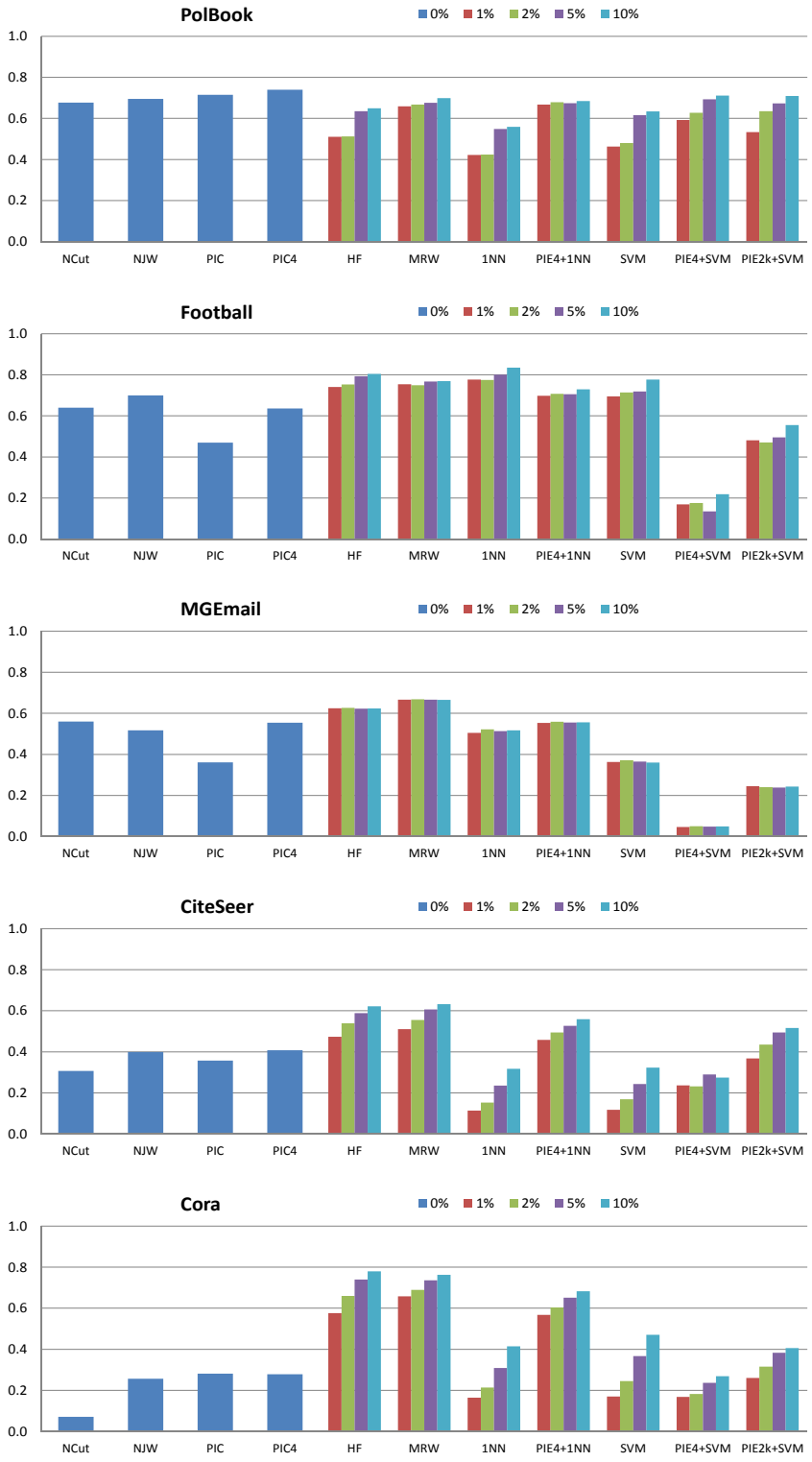



Figure 4.6: F1 comparison between SSL methods and clustering methods on network data. Colors correspond to the percentage of data used as training/seed instances; 0% corresponds to a clustering problem.

# Chapter 5

# MRW's Relation to Other Methods

Broadly, both MultiRankWalk (MRW) and the harmonic fields method (HF) falls under a category of graph-based SSL methods that use *label propagation* to determine which class an instance (a node in the graph) belongs to. More specifically, MRW and HF are *probabilistic* label propagation SSL methods—by which we mean the ways they propagate labels are stochastic processes and the resulting classification confidence scores are probabilities. They can be further distinguished from other probabilistic label propagation methods in that they are both related to Markov random walks on graphs.

## 5.1 RWR Revisited

As discussed in Section 4.2, MRW uses random walk with restart (RWR) scores to classify unlabeled nodes in a graph. Random walk scores are the stationary distribution of the Markov chain induced from the column-stochastic transition matrix $P$, which is derived from the affinity matrix $A$ of the instances in dataset $\mathcal{X}$. The stationary distribution is represented by the vector $\boldsymbol{\nu}$ in the following equation:

$$\boldsymbol{\nu} = (1 - d)\mathbf{r} + dP\boldsymbol{\nu} \tag{5.1}$$

where $\mathbf{r}$ is a normalized restart vector where $|\mathbf{r}| = n$ and $\|\mathbf{r}\|_1 = 1$, and $d$ is the damping factor. We can solve for $\boldsymbol{\nu}$ analytically:

$$\begin{aligned}
\boldsymbol{\nu} &= (1-d)\mathbf{r} + dP\boldsymbol{\nu} \\
\boldsymbol{\nu} - dP\boldsymbol{\nu} &= (1-d)\mathbf{r} \\
(I - dP)\boldsymbol{\nu} &= (1-d)\mathbf{r}
\end{aligned}$$

and since $I - dP = (I - dP^T)^T$, if $I - dP^T$ is strictly diagonally dominant[1] and therefore invertible, $I - dP$ is also invertible, giving us

$$\boldsymbol{\nu} = (1-d)(I - dP)^{-1}\mathbf{r}$$

which allows us to solve for $\boldsymbol{\nu}$ given $\mathbf{r}$ once we have $(I - dP)^{-1}$. This matrix inversion, however, is computationally infeasible if $n$, the number of instances, is large[2], so we approximate $\mathbf{v}$ iteratively with the power method:

$$\boldsymbol{\nu}^{t+1} = (1-d)\mathbf{r} + dP\boldsymbol{\nu}^t$$

until $\boldsymbol{\nu}^t$ converges. Here we show $\boldsymbol{\nu}^t$ converges to $\boldsymbol{\nu}$:

$$\begin{aligned}
\boldsymbol{\nu}^t &= (1-d)\mathbf{r} + dP\boldsymbol{\nu}^{t-1} \\
\boldsymbol{\nu}^t &= (1-d)\mathbf{r} + dP((1-d)\mathbf{r} + dP\boldsymbol{\nu}^{t-2}) \\
\boldsymbol{\nu}^t &= (1-d)\mathbf{r} + dP((1-d)\mathbf{r} + dP((1-d)\mathbf{r} + dP\boldsymbol{\nu}^{t-3})) \\
&\dots \\
\boldsymbol{\nu}^t &= (1-d)\sum_{i=0}^{t-1}(dP)^i\mathbf{r} + (dP)^{t-1}\boldsymbol{\nu}^0
\end{aligned}$$

[1] A matrix $A$ is *diagonally dominant* if $|A_{ii}| \geq \sum_{j \neq i} |A_{ij}|$ for all $i$, and it is *strictly diagonally dominant* if $|A_{ii}| > \sum_{j \neq i} |A_{ij}|$ for all $i$.

[2] In general, $O(n^2)$ space and $O(n^3)$ time.

and given that $0 < d < 1$ and the eigenvalues of $P$ are in $[-1, 1]$ (due to the Perron-Frobenius theorem), we have

$$\lim_{t \to \infty} (dP)^{t-1} \boldsymbol{v}^0 = \mathbf{0}$$

and[3]

$$\lim_{t \to \infty} \sum_{i=0}^{t-1} (dP)^i = (I - dP)^{-1}$$

which gives us

$$\lim_{t \to \infty} \boldsymbol{v}^t = (1 - d)(I - dP)^{-1} \mathbf{r}$$

$$\lim_{t \to \infty} \boldsymbol{v}^t = \boldsymbol{v}$$

Note that the result of the iterative algorithm does not depend on the initial vector $\boldsymbol{v}^0$; $\boldsymbol{v}^t$ will converge to the same result as long as $\boldsymbol{v}^0$ is nonzero. In practice, though, often $\boldsymbol{v}^0$ is set to $\mathbf{r}$ so the iterative process simulates a spreading of labels from the seed nodes.

## 5.2 Steady-state Conditions for RWR

It is important to note that the RWR results in the previous section (5.1) are based on the following assumptions:

1. There exists a unique stationary distribution $\boldsymbol{v}$.
2. $I - dP^T$ is a strictly diagonally dominant matrix.

[3] See Appendix D.1 for detailed derivation.

Due to the Ergodic theorem, the first condition is met if the Markov chain underlying Equation 5.1 is *aperiodic*[4] and *irreducible*[5] [44, 76]. Both aperiodicity and irreducibility can be achieved by adding an edge with a relatively small weight between all pairs of nodes, effectively making $A$ a complete graph. Instead of modifying $A$, we can simply make sure all elements of $\mathbf{r}$ are positive—this has the same effect as adding *teleportation* as in [44, 82]. The second condition is met simply when $d$ satisfies $0 < d < 1$, since each row of $P^T$ sums to 1.[6] Teleportation is an efficient way to augment $A$ to be a complete graph to ensure a unique stationary distribution and is widely used in ranking algorithms [44, 82]; however, in the context of SSL, it may not be necessary.

In the case that the Markov chain is reducible—which implies that the network underlying $A$ has more than one strongly connected component—$\boldsymbol{\nu}$ may reach a stationary distribution, but such a distribution may not be unique. In the context of SSL classification, this can be "fixed" by considering each component as its own learning problem. This is an obvious solution since seed labels cannot propagate across disconnected components, nor should they. By definition, this solution also ensures each subproblem has an irreducible Markov chain.

In the case that the Markov chain is periodic[7] (e.g., a bipartite graph), all we need to do is augment all the nodes in the graph with a self-edge with a relatively small weight to ensure aperiodicity (since the smallest cycle is of length 1). In fact, in the context of SSL using RWR, no augmentation is necessary as long as the restart vector $\mathbf{r}$ is nonzero, since a nonzero $\mathbf{r}$ implies the existence of at least one self-edge.

[4] A Markov chain is *aperiodic* if its periodicity is 1, otherwise it is *periodic*. A state $i$ has period $k$ if $k$ is the greatest common divisor of all possible number of steps it takes to return to state $i$. The Markov chain has periodicity of $k$ if $k$ is the greatest common divisor of all its states' periods.

[5] A Markov chain is *irreducible* if it is possible to get from any state to any other state.

[6] Note that this holds true whether $P^T_{ii} = 0$ or $P^T_{ii} > 0$.

[7] In general, aperiodicity is a very mild condition for "natural networks"; i.e., most of them, such as friendship networks or email networks, are aperiodic.

## 5.3 MRW Related SSL Methods

SSL classification methods closely related to MRW have been proposed previously. Here we will briefly describe and compare some of the most popular ones.

### 5.3.1 Learning with Local and Global Consistency

The local and global consistency method (LGC) [111] is a semi-supervised classification method with an iterative algorithm with the following regularization framework:

$$Q(F) = \frac{1}{2}\left(\sum_{i,j=1}^{n} A_{ij}\left\|\frac{1}{\sqrt{D_{ii}}}F_i - \frac{1}{\sqrt{D_{ii}}}F_j\right\|^2 + \mu\sum_{i=1}^{n}\|F_i - Y_i\|^2\right)$$

where $F_{ij}$ contains the score for instance $i$ with respect to class $j$ and $Y_{ij} = 1$ if instance $i$ is a seed labeled with class $j$, otherwise $Y_{ij} = 0$. The class for an unlabeled instance $i$ is determined by $argmax_j F_{ij}$. $F$ can be calculated by iterating:

$$F^{t+1} = (1-\alpha)Y + \alpha S F^t$$

where $\alpha = \frac{1}{1+\mu}$ and $S = D^{-1/2}AD^{-1/2}$. Two variations of this method are also proposed where $S = D^{-1}A$ and $S = AD^{-1}$, respectively. In fact, the second variation (LGCv2) is equivalent to MRW, given that the affinity matrix $A$ is also equivalent. We differentiate our work from LGCv2 in the following respects:

- LGCv2 is only briefly described as a variation on a regularization framework for SSL, with no mention of connection to random walks on graphs.
- The $A$ associated with LGC is based on the Gaussian similarity kernel between feature vectors, whereas we consider more general affinity matrices.
- No experiments were done on network datasets.
- No experiments were done on seeding preference.

### 5.3.2 Web Content Classification Using Link Information

In [42], RWR is also used for classifying websites using links between pages. In this work (which we will abbreviate as WCC), information from class-based RWR is used to generate a "profile" (a feature vector) for each instance, and this profile, optionally combined with other sources of information, is the input to binary SVM classifiers that output the final classification prediction. Experiments are done on a Yahoo! web graph using the Open Directory for labels. In contrast, MRW uses RWR probabilities directly for classification and we verify MRW's effectiveness on a variety of network and text classification tasks.

Table 5.1 is a summary of the above RWR-based SSL classification methods.

| Work | View | Classify | Application |
|---|---|---|---|
| LGCv2 [111] | label consistency regularization | max | image, text |
| WCC [42] | RW scores as features | SVM | websites |
| MRW [65] | seed proximity via RW | max | blog & citation networks |

Table 5.1: A summary of RWR-based SSL classification methods with equivalent iterative update algorithm. *View* is how the method is motivated; *Classify* is to how class labels are assigned after the iterative algorithm converges at a stable per-class distribution; *Application* is the type of data on which the method is used.

## 5.4 Connections between HF and MRW

The harmonic fields method (HF) [114] can also be viewed as a probabilistic label propagation SSL method related to random walks on graph. However, an important distinction must be made—that while the methods discussed in Section 5.3 are related to *forward random walks*, HF is related to *backward random walks*.

The main strategy of HF is to first compute a real-valued function $f : \mathcal{X} \rightarrow \Re$ with respect to $A$ with certain nice properties, and then assign labels based on $f$. Labeled instances $x \in \mathcal{X}^L$ are constrained to take on values based on their labels (e.g., for binary

classification $y \in \{0, 1\}$, a positive labeled instance $i$ would have $f(i) = 1$ and a negative labeled instance $j$ would have $f(j) = 0$). After $f$ is computed, labels can be assigned with a simple decision function, such as the following, in the case of binary classification:

$$y_i = \begin{cases} + & \text{if } f(i) > \frac{1}{2} \\ - & \text{otherwise} \end{cases}$$

where $y_i$ is the predicted label of instance $i$ and $+$ is the positive label and $-$ the negative label. A desirable property for $f$ would be that *similar* instances (e.g., nearly points in Euclidean space) would have similar labels, thus motivating the quadratic energy function:

$$E(f) = \frac{1}{2} \sum_{i,j} A_{ij}(f(i) - f(j))^2$$

The minimum energy function $f = \operatorname{argmin}_{f|y_L} E(f)$ is *harmonic*, meaning that the values of $f$ for each unlabeled instance is the weighted average of $f$ for its neighbors:

$$f(i) = \frac{1}{D_{ii}} \sum_{j} A_{ij} f(j)$$

where $D_{ii} = \sum_j A_{ij}$ and $i$ indexes over $X^u$. It turns out that $f$ is unique and can be simply expressed as [114]:

$$f = Wf$$

where $W = D^{-1}A$. Therefore $W$ is the row-stochastic transition matrix of $A$. Recall that in Section 5.1, the random walk transition matrix is $P = AD^{-1}$; so it follows that $W = P^T$ and $P = W^T$. This shows there is a close relationship between MRW and HF. Actually, since MRW is a label propagation method and $P^t$ computes the transition probabilities

if we run the random walk $t$ steps forward, it follows that $W^t$ computes the transition probabilities if we run the random walk $t$ steps *backward*.

With just the transition matrix ($P = AD^{-1}$ or $W = D^{-1}A$), neither forward random walks (FRW) nor backward random walks (BRW) is useful as a SSL method. FRW is made useful for label propagation by incorporating label-biased restart probabilities; and using a random walk with restart (RWR) per labeled class allows us to learn from labeled and unlabeled data, assuming that the similarity function represented by $A$ captures the underlying homophily of the data. BRW, on the other hand, is made useful for label propagation by turning labeled instances into *sink nodes*.

If we view the affinity matrix $A$ as a graph, where the nodes correspond to data instances and edges are weighted according to $A$, and designate nodes corresponding to instances in $\mathcal{X}^L$ as sink nodes, then we can define a random walk process on this graph as mentioned in Section 5.1, except that a walk terminates when it reaches a sink node. These sink nodes are called *absorbing states* in the Markov chain corresponding to such process, and the probability of a random walk ending up at an absorbing state is the walk's *absorption probability* by the state.

The Markov chain defined by the transition matrix $P$ is *reversible*, allowing us calculate the random walk probabilities in reverse given an "ending distribution". FRW answers the question "given the current probability distribution over the states, what is the most likely distribution $t$ time steps into the future?", and BRW answers the question "given the current probability distribution over the states, what is the most likely distribution $t$ time steps ago?" In other words, BRW with sink nodes allows us to compute the probability that a random walker starting at a node (or state) $a$ will end up at a sink node (or absorbing state) $b$, as opposed to another sink node (or absorbing state).

**Class Mass Normalization**

Class mass normalization (CMN) is a heuristic for adjusting the class decision function $f$ [114]; here we discuss it briefly because CMN is often recommended when using HF. The idea is, given that we have prior knowledge (or an accurate estimate) of the distribution of class labels for the unlabeled instances, we can adjust $f$ so that the overall probability mass matches the prior distribution. In the case of binary classification, and we are able to obtain prior class distribution $q$ and $1-q$, then the decision function would be:

$$y_i = \begin{cases} + & \text{if } q\frac{f(i)}{\sum_i f(i)} > (1-q)\frac{1-f(i)}{\sum_i 1-f(i)} \\ - & \text{otherwise} \end{cases}$$

Note that the above decision function adjusts for the class mass counting both labeled and unlabeled instances; if for some reason the labeled instances have a different distribution from the class prior, CMN should be done on just the unlabeled instances. While CMN could be useful when the data has a skewed class distribution, here are a few items to consider:

- CMN is a heuristic that is applied apart from the HF objective function.
- In a non-experimental setting the prior class distribution may be difficult to estimate.
- When resources only allow a few labeled instances, it is difficult to accurately estimate a class prior; it may be of more importance to choose carefully *which* instances to label, as shown in Section 4.4.3.
- It is not clear how to use CMN in a setting where an instance may have multiple labels.

#### Learning the Similarity Function

The harmonic functions methods was original proposed to be used with the Gaussian similarity function where the affinity would be defined as:

$$A_{ij} = exp\left(-\sum_{d} \frac{(x_{id} - x_{jd})^2}{\sigma_d^2}\right)$$

Note that this particular definition has an anisotropic (different in different dimensions) bandwidth parameter $\sigma$, as opposed to the more common isotropic one. An gradient descent method was also proposed with HF to learn $\sigma$ from data [114]. However, this method is only applicable to the Gaussian similarity function and does not scale to large datasets.[8] Learning parameters for a manifold for various computation applications is itself a growing field of study; in this work we will focus on cases where the affinity matrix $A$ is a given or can be obtained without complex learning methods.

## 5.5 HF Related SSL Methods

### 5.5.1 Weighted-voted Relational Neighbor Classifier

The weighted-voted relational neighbor classifier (wvRN) is one of the methods presented in a networked data classifier framework [71]. Though it is one of the simplest in the framework, it is also one of the most accurate on a number of benchmark datasets and therefore recommended as a strong baseline. Although it turns out that wvRN and HF are equivalent [71], wvRN is proposed as a collection of local classifiers with an iterative solution, whereas HF is proposed as a global optimization problem with a linear solution. Of course, the equivalence between the two solutions means that HF can be more efficiently solved using an iterative algorithm like wvRN if $A$ is sparse.

[8] In general, a Gaussian similarity function do not scale to large datasets due to its $O(n^2)$ time and space complexity when constructing $A$.

### 5.5.2 The Rendezvous Algorithm

The Rendezvous algorithm [9] associates each node with a particle that moves between nodes according to $W$. As with the BRW SSL view described, labeled nodes are set to be absorbing states of the Markov random walk, and the probability of each particle to be absorbed by the different labeled nodes is used to derive a distribution over the associated missing label. The difference between this method and HF [114] and wvRN [71] are noted below:

- After $A$ is computed using the appropriate similarity function, a sparse version, $\hat{A}$, is derived using a $k$-nearest neighbor algorithm (kNN).
- Since kNN does not produce symmetric results ($k$-nearest neighbors of $a$ may not include $b$ even if $a$ is a nearest neighbor of $b$), $\hat{A}$ is not necessarily symmetric.
- The solution is obtained via eigendecomposition.

### 5.5.3 Adsorption

Adsorption [13,93] uses an iterative method similar to that of wvRN, with the exception that, instead of constraining the labeled nodes to have a probability of 1 (or a maximum score) matching their assigned labels, it instead uses *shadow nodes* to "inject" the assigned labels into the propagation process. Only a labeled node has a shadow node, and a shadow node has only one edge connecting it to its corresponding labeled node. In each iteration, only the shadow nodes are constrained to their assigned values, so even the labeled nodes will allow labels other than their assigned label to "propagate through" or take on a label other than their originally assigned label (though unlikely). Shadow nodes are very similar to the "dongle" nodes proposed in the original HF paper for incorporating external classifiers [114]. Shadow nodes or dongle nodes may be useful where there are noisy labels.

| Work | View | Algorithm | Similarity |
|---|---|---|---|
| HF [114] | harmonic energy minimization | gradient descent | Gaussian |
| wvRN [71] | consistent networked classifiers | iterative | network links |
| Rendezvous [9] | random walk+absorbing states | eigendecomposition | Gaussian+kNN |
| Adsorption [13] | iterative averaging | iterative | network links |

Table 5.2: A summary comparison of backward random walk (BRW) SSL classification methods. In general all of these methods can be solved using a similar (though not entirely the same) iterative method. *View* refers how the method is motivated; *Algorithm* refers to the core algorithm used to derive the solution; *Similiary* refers to the similarity function used in constructing the affinity matrix $A$.

We summarize the comparison of the above mentioned BRW-related methods in Table 5.2, which shows how the methods differ in motivation, similarity function, and implementation. For a formal comparison of these methods we refer to Koutra et al. [56], which shows that many of these methods are inversions of closely related matrices.

## 5.6 Discussion

We note that PIC (Chapter 2), MRW, and HF as defined in Section 5.4 all use a static, predefined pair-wise similarity function that specifies the input affinity matrix. In other words, the notion of "distance" between different clusters or between labeled and unlabeled data points is fixed, and these methods assume the fixed distances will result in a good classification solution or a good clustering. Though for the scope of this thesis we assume this is the case, an adaptively learned distance or similarity function may result in a better solution, especially if the similarity is a function of different features or edge types (e.g., a social network where the each edge is labeled with a relation type such as "friend", "family", and "co-worker").

For example, if we have node-level features for the input network, instead of taking the edge weights "as is", we can learn a weight function, parameterized by a weight vector over the similarity of node-level features between neighbors and optimized based on the RWR scores (the notion of distance behind MRW) of the training data [10]. Another

example is learning the bandwidth parameter $\sigma$ of the Gaussian kernel similarity function for the harmonic functions method [114]. We also propose an adaptively learned version of PIC as future work in Section 9.5 when some supervision is available in the form of clustering constraints.

# Chapter 6

# Implicit Manifolds

## 6.1 Introduction

PIC provides a simple and scalable alternative to spectral clustering methods for clustering large network datasets; and graph-based semi-supervised learning methods such as MRW have shown to be efficient and effective on network data by propagating labels to neighboring nodes. They are well-suited for large datasets with "natural" graph structures, such as a network of hyperlinked political blogs [64] or a YouTube video dataset where nodes represent videos and weighted edges between nodes represent how often the same users view these two videos [13]. These methods are efficient for large datasets because their computational complexities are linear in the number of edges in the graph, and most large natural graphs are sparse in the number of edges.

For other types of data that do not come naturally as a graph, these methods can also be applied by constructing a graph where the nodes are the instances and the edges are weighted by the pair-wise similarity between the instances. However, whereas a natural network or a natural graph is often sparse, a graph of pairwise similarities between instances can be dense.

A classic example of such type of data is natural language text in the form of a "bag of words" (BOW) representation. BOW is a simple but intuitive way to represent the meaning of a document by its word tokens, without additional linguistic or semantic structure. BOW is popular for information retrieval (IR) [72] and text categorization tasks [62] due to its simplicity and scalability. To use many graph-based clustering or SSL methods, we first need to construct a manifold in the form of a pair-wise affinity matrix—a graph where the nodes are documents and weighted edges denote word similarity between documents. An important observation is that while documents themselves can be represented efficiently (e.g., a sparse $n \times m$ matrix where $n$ is the number of documents and $m$ is the size of the word dictionary), the affinity matrix $A$ will be rather dense ($\sim n^2$ edges) because most document pairs will share at least a few common words, which means non-zero similarity, resulting in a dense affinity matrix.

With dense matrices many graph-based methods are no longer scalable. To construct, to store, and to operate on such a graph is at least $O(n^2)$ in terms of both time and space. In order to make clustering and semi-supervised learning methods practical to general data, prior work has mostly relied on *explicitly sparsifying* the dense affinity matrix in one of two ways:

1. Sample the data (i.e., sample the nodes and/or the edges) and do computation on a much smaller matrix [22, 38, 107, 110].
2. Construct a much smaller or sparser graph that represents the original large dense graph [22, 31, 68, 103]. A common method is constructing a k-nearest neighbor graph, where a node is connected to k other nodes that are most similar to it.

A drawback of sampling and sparsifying methods is that they gain speed and storage efficiency at the cost of losing some pair-wise similarity information available in the data. In addition, the construction of these sparsified graphs incurs additional non-trivial computation costs, and often expertise and familiarity with the sparsifying technique is required in order to apply it properly. These techniques have in common that all of them

they still use the same core algorithm; ultimately, a similarity matrix is computed and stored, and a rather expensive computation like eigendecomposition is performed.

In contrast, we proposed a simple, practical, and general technique and framework, called *implicit manifolds* (IM) [66, 67], under which particular similarity functions and learning algorithms can provide solutions that are *exactly equivalent* to using a *complete pair-wise similarity manifold* while keeping runtime and storage linear to the input size; and this is done without sampling or calculating a sparse representation of the data.

This is possible based on a simple observation. If a graph-based clustering or an SSL method has at its core iterated matrix-vector multiplications, where the matrix is based on the adjacency matrix of the graph, then if the graph is sparse (number of edges $|E| = O(n)$), the method terminates quickly and requires a relatively small amount of memory; in other words, sparse matrix-vector multiplication is cheap. If the graph is dense (e.g., $|E| = O(n^2)$), the method will be slow and require a large amount of memory; in other words, dense matrix-vector multiplication is expensive. So if we are able to decompose the dense matrix into a number of sparse matrices, the dense matrix-vector multiplication becomes a series of sparse matrix-vector multiplications, reducing the cost of both space and time to linear w.r.t. input size. Because the pair-wise similarity manifold represented by the affinity matrix is never constructed explicitly, we call this general technique *implicit manifold construction*.

## 6.2 Path-Folding

The idea of path-folding (PF) is often used in network science to transform a two-mode network (a graph with two distinct types of nodes) into a one-mode network. PF is related to the notion of a bipartite graph. A bipartite graph consists of two mutually exclusive sets of nodes where only edges between nodes of different groups are allowed. Any dataset with instances and features can be viewed as a bipartite graph, where one

set of nodes corresponds to instances and the other set corresponds to features. If an instance has a certain feature, an edge exists between the instance node and the feature node; if the feature is numerical or if it is weighted, the edge can be weighted accordingly. If two instances contain the same feature, a path of length two exists between them. If two instances are very similar (i.e., they share many features), there will be many paths of length two between them; if two instances are very dissimilar, then there would be very few such paths or none at all. Thus the number of paths (and their weights) between two instance nodes in this graph can be viewed as a similarity measure between two instances.

If we are just interested in the similarity between instances, we may "fold" the paths by counting all paths of length two between two instances and replacing the paths with a single edge, weighted by the path count. This "folding" can be expressed concisely with a matrix multiplication:

$$A = FF^T$$

where rows of $F$ represent instances and columns of $F$ represent features. $A$ is then the "folded" graph—each node is an instance, and a weighted edge between two instances ($A_{ij}$) represents the count of all paths of length two in the original "unfolded" graph $F$.

Now consider the density of these two different representations, the "unfolded" bipartite graph $F$ and the "folded" graph $A$, as the size of the dataset (the number of instances) grows. In real datasets, $F$ is often considered sparse because either (a) the feature space is small w.r.t. dataset size (e.g., census data with a small set of per-household questions) or (b) the feature space is large but each instance has only a small fraction of these features (e.g., document data with word features). $A$, on the other hand, is likely dense.

In the context of large text datasets. $F$ will most certainly be a sparse matrix; there are a large number of words in the vocabulary, but only a very small fraction of them will occur in any single document. $A$ is quite the opposite. $A_{ij}$ is zero only if no words are shared between documents $i$ and $j$; yet the very skewed distribution of word occurrences [72], and in particular that of the most common words, makes $A_{ij}$ highly likely to be non-zero, which subsequently makes $A$ dense. As the number of documents increases, $A$, an explicit representation of document similarity, becomes very costly in terms of storage and processing time.[1] What we want to do is to *not* calculate $A$ and use its "unfolded" form $FF^T$ on-the-fly for the matrix-vector multiplication operations—the core of iterative clustering and SSL methods such as PIC, MRW, and HF (Sections 6.4 and 6.5).

Before using the affinity matrix in its "unfolded" form, we need to do one more thing. Recall that $W$ and $P$ are the normalized forms of $A$ where $W = D^{-1}A$ and $P = AD^{-1}$. We need to find the diagonal matrix $D^{-1}$ without constructing $A$. It turns that the values of the diagonal matrix $D^{-1}$ can also be calculated efficiently via a number of sparse matrix-vector multiplications using the same idea that gave rise to IM. Since we can calculate a vector $\mathbf{d}_i = \sum_j A_{ij}$ of row sums by

$$\mathbf{d} = A\mathbf{1}$$

where $\mathbf{1}$ is the constant vector of all 1's, then

$$\mathbf{d} = FF^T\mathbf{1}$$

[1] Besides text data, many other types of data display a similar behavior. For example, a census dataset of a large population would have a small number of features per individual (because the feature space is small), but would result in a large and dense affinity matrix.

Then we can construct the matrix D by setting its diagonal to $\mathbf{d}$: $D_{ii} = \mathbf{d}(i)$. Now the normalized matrices $W$ and $P$ become:

$$W = D^{-1}(F(F^T))$$
$$P = F(F^T(D^{-1}))$$

As we will see later, the parentheses specifying the order of operations is vital in making them efficient series of sparse matrix-vector multiplications.

## 6.3 Implicit Manifolds for Text Data

If instead of a bipartite graph we view the rows of $F$ as feature vectors of documents in vector space, then "path-folding" is equivalent to the *inner product similarity* of documents in a document *vector space model*, often used in information retrieval problems [72]. However, this is just one of many similarity functions often used for measuring document similarity. For example, one may want to normalize the document feature vectors by their lengths.

### 6.3.1 Inner Product Similarity

As mentioned in Section 6.2, the inner product similarity simply decomposes the affinity matrix as:

$$A = FF^T \tag{6.1}$$

### 6.3.2 Cosine Similarity

It turns out that the implicit manifold is readily applicable to other similarity functions, *as long as the manifold can be represented with a series of sparse matrix multiplications*. Here

we consider *cosine similarity* [62,72], widely used for comparing document similarity:

$$\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|} \tag{6.2}$$

Here $\cos(\mathbf{a}, \mathbf{b})$ is the cosine of the angle between vectors $\mathbf{a}$ and $\mathbf{b}$. For the normalizing term $1/(\|\mathbf{a}\| \, \|\mathbf{b}\|)$ we calculate an additional diagonal matrix $N_{ii} = 1/\sqrt{(F_i F_i^T)}$ where $F_i$ is the $i$th row-vector of $F$, resulting in the following decomposition for $A$:

$$A = NFF^TN \tag{6.3}$$

Following inner product similarity the values of the diagonal matrix $D$ can be computed by $\mathbf{d} = NFF^TN\mathbf{1}$. All operations in constructing $N$ and $D$ are sparse matrix-vector multiplications.[2]

### 6.3.3 Bipartite Graph Walk Similarity

Another useful similarity measure is based on bipartite graphs. Unlike the inner product similarity where a bipartite graph is "folded" into a unipartite graph, here we are interested in simulating a Markov random walk on a bipartite graph.

For methods that correspond to forward random walks such as MRW, $P$ is defined here as

$$P = FC^{-1}F^TR^{-1} \tag{6.4}$$

where $R$ is the diagonal matrix containing the row sums of $F$ such that $R_{ii} = \sum_j F_{ij}$, and $C$ is the diagonal matrix containing the column sums of $F$ such that $C_{ii} = \sum_i F_{ij}$.

[2]While we could preprocess $F$ to be cosine-normalized and consequently simplify the above to a inner product similarity, we point out that with large datasets it may be inefficient to store a different version of the dataset for every similarity function one might want to use; calculating similarity functions on-the-fly may prove to be a more efficient approach.

For methods that correspond to a backward random walks such as PIC and HF, $W$ is defined as

$$W = \mathsf{R}^{-1}\mathsf{F}\mathsf{C}^{-1}\mathsf{F}^{\mathsf{T}} \tag{6.5}$$

where $\mathsf{R}$ and $\mathsf{C}$ are defined as above. Note that unlike inner product similarity or cosine similarity, for bipartite graph walk manifold $W \neq \mathsf{P}'$. However, if here we define $\mathsf{A} = \mathsf{F}\mathsf{C}^{-1}\mathsf{F}^{\mathsf{T}}$, then $W = \mathsf{R}^{-1}\mathsf{A}$ and $\mathsf{P} = \mathsf{A}\mathsf{R}^{-1}$—the two are still closely related. Note also that it is unnecessary to calculate $\mathsf{D}^{-1}$ here since $W$ and $\mathsf{P}$ are already properly normalized transition matrices.

The bipartite similarity manifold can be interpreted in two ways. First, like its namesake, it simulates a random walk on $\mathsf{F}$ as a bipartite graph, where each iteration is equivalent to taking two random steps—the first step walks from the instances to the features, and the second step walks from the features back to the instances. It can also be interpreted as a inner product similarity with the features re-weighted inversely proportional to their dataset frequency. This is closely related to the tf-idf weighting scheme found in document retrieval literatures [72].

## 6.4 Implicit Manifolds for Clustering

At the core of PIC is a simple calculation: a matrix-vector multiplication $W\boldsymbol{v}^{\mathrm{t}}$, which readily fits into the implicit manifold framework: If we decompose the matrix $W$ into a series of matrix multiplications, the original PIC matrix-vector multiplication becomes a series of matrix-vector multiplications. To use implicit manifold for PIC, we simply replace $W$ with the appropriate manifold construction. For example, to use cosine simi-

larity, the $W\boldsymbol{v}^t$ in Figure 2 becomes:

$$D^{-1}(N(F(F^T(N\boldsymbol{v}^t)))) \tag{6.6}$$

where N and $D^{-1}$ are defined and calculated as in Section 6.3.2. The parentheses above emphasize again that the order of operations is vital in preserving the computational efficiency of the algorithm.

In this section we present evaluation results for the implicit manifolds version of PIC on a document clustering task [66], showing clustering performance similar to that of spectral clustering using a full affinity matrix; this is achieved without incurring the cost, time-wise and space-wise, of constructing and operating on a dense similarity matrix.

### 6.4.1 Accuracy Results

We choose the RCV1 text categorization collection [62] as the dataset for the document classification task. RCV1 is a well-known benchmark collection of 804,414 newswire stories labeled using three sets of controlled vocabularies. We use the test split of 781,256 documents and category labels from the *industries* vocabulary. To aid in clustering evaluation, documents with multiple labels and categories with less than 500 instances were removed, following previous work [22]. We ended up with 193,844 documents and 103 categories.

We generate 100 random category pairs and pool documents from each pair to create 100 two-cluster datasets: first, we randomly draw a category from the 103 categories—this is category A. Then for candidates of category B, we filter out category A itself and any other category that is more than twice the size or less than half the size of category A. Finally, category B is randomly drawn from the remaining categories. This whole process is repeated 100 times. The filtering is done so we do not have datasets that are overly skewed in cluster size ratio, leading to accuracy results that are difficult to

interpret. Since the *industries* vocabulary supports many fine distinctions, we end up with 100 datasets of varying difficulty.

Each document is represented as a log-transformed if-idf (term-frequency $\cdot$ inverse document-frequency) vector, as is typically done in the information retrieval community for comparing similarity between documents [62,72].

We compare PIC against two methods—the standard k-means algorithm and Normalized Cuts [91]. We compare results with two versions of Normalized Cuts: $\text{NCut}_{\text{EVD}}$ and $\text{NCut}_{\text{IRAM}}$. $\text{NCut}_{\text{EVD}}$ uses the slower but more accurate classic eigenvalue decomposition for finding eigenvectors. $\text{NCut}_{\text{IRAM}}$ uses the fast *implicitly restarted Arnoldi method* [60], a more memory-efficient version of the Lanczos algorithm that produces approximations to the top or bottom eigenvectors of a square matrix.

In this experiment we use PIC modified with the cosine similarity function, with $0.00001/n$ as the convergence threshold, where $n$ is the number of documents, and with random initial vectors where components are randomly drawn from [0,1). For both PIC and the NCut methods, we run k-means 10 times on the embedding and choose the result with the smallest WCSS as the final clustering.

We evaluate the clustering results according to the *industries* category labels using two metrics: *accuracy* and *normalized mutual information* (NMI).

Accuracy in general is defined to be the percentage of correctly labeled instances out of all the labeled instances. Clustering accuracy here is the best accuracy obtainable by a clustering if we were to assign each cluster a unique category label by consider all such possible assignments and then pick one that maximizes the labeling accuracy. NMI is a information-theoretical measure where the mutual information of the true labeling and the clustering are normalized by their entropies. See Appendix A.2 for definitions of these metrics.

The experimental results are summarized in Table 6.1, showing the accuracy and NMI for the methods compared, averaged over 100 category pair datasets. The "baseline"

number for accuracy is the average accuracy of a trivial clustering where all the data points are in one cluster and none in the other (i.e., the accuracy of having no clusters). This is provided due to the tendency for clustering accuracy to appear better than it actually is. The differences between numbers in Table 6.1 are all statistically significant *with the exception* of those between $\text{NCut}_{\text{EVD}}$ and PIC, where the p-values of one-tailed paired t-tests of accuracy and NMI are 0.11 and 0.09 respectively.

| | Accuracy | NMI |
|---|---|---|
| baseline | 57.6 | - |
| k-means | 69.4 | 0.263 |
| $\text{NCut}_{\text{EVD}}$ | 77.6 | 0.396 |
| $\text{NCut}_{\text{IRAM}}$ | 61.6 | 0.094 |
| PIC | 76.7 | 0.382 |

Table 6.1: Summary of clustering results. Higher numbers indicate better clustering performance according to ground truth labels. All differences are statistically significant with the exception of those between $\text{NCut}_{\text{EVD}}$ and PIC.

The accuracy values correlate with those of NMI, and $\text{NCut}_{\text{EVD}}$ is the most accurate algorithm, though not statistically significantly more so than PIC. Both $\text{NCut}_{\text{EVD}}$ and PIC do much better than k-means, a typical result in most prior work comparing k-means and methods using pair-wise similarity [22, 80, 103]. We are surprised to find $\text{NCut}_{\text{IRAM}}$ doing much worse than all other methods including k-means; the degree to which it failed the task is even more pronounced in NMI, showing the clustering is close to random. In prior work [22, 60] and in our previous experience with other datasets $\text{NCut}_{\text{IRAM}}$ usually does as well or nearly as well as $\text{NCut}_{\text{EVD}}$. Perhaps a more advanced tuning of the parameters of IRAM is required for better approximations to eigenvectors. Regardless, the conclusions we draw from these experiments is no less significant even if $\text{NCut}_{\text{IRAM}}$ were to perform just as well as $\text{NCut}_{\text{EVD}}$.

Since the datasets are of varying difficulties, we are interested in how well PIC performs compared to other methods on the individual datasets. So we plot the accuracy of other methods against that of PIC in Figure 6.1. The x-axes correspond to PIC accuracy

and the y-axes correspond to that of the compared method. A point above the diagonal line indicates a "win" for PIC over the compared method for a category pair and a point below indicates otherwise. A point on the diagonal line corresponds to a "tie".

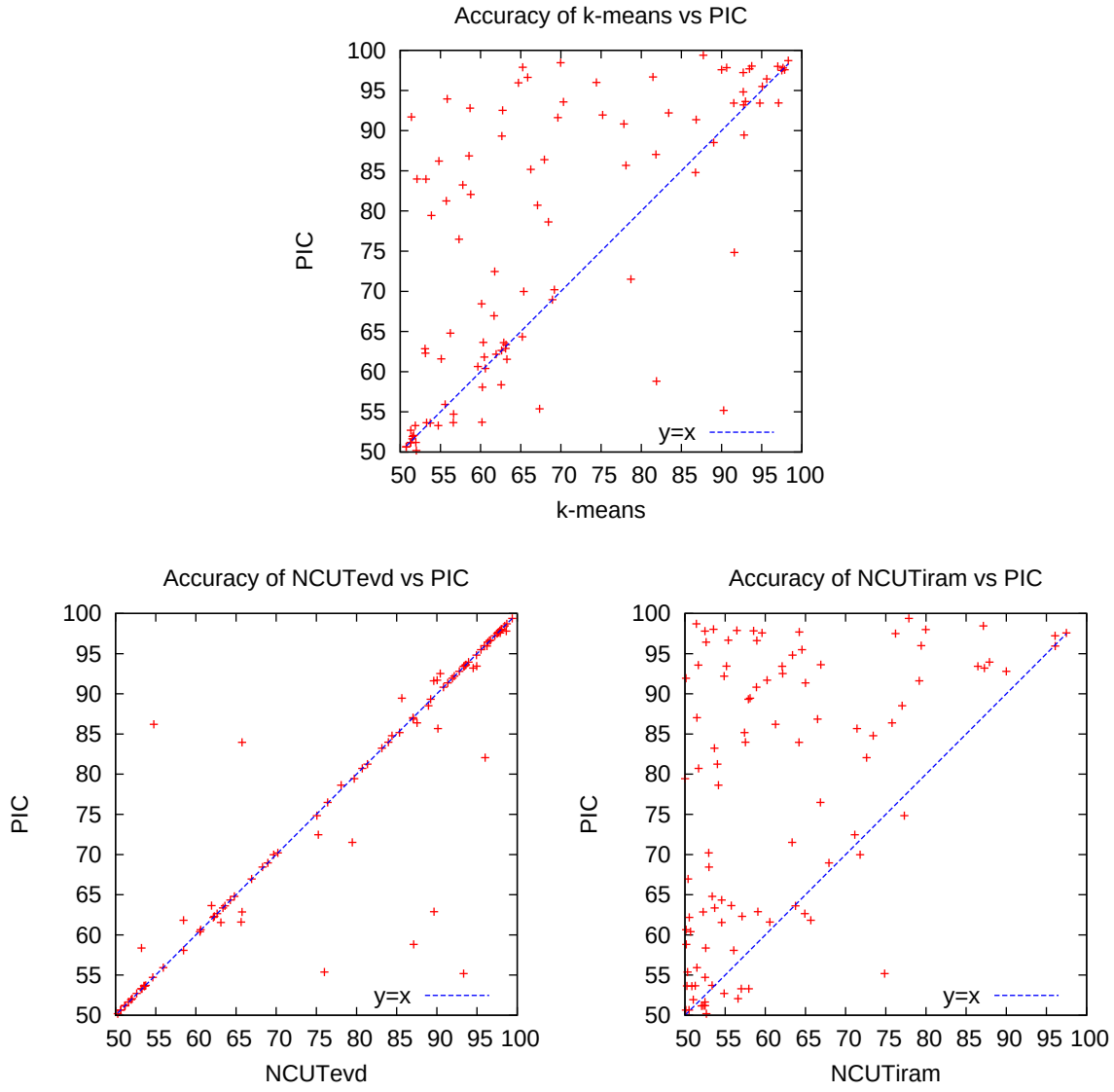



Figure 6.1: Clustering accuracy correlation plots between PIC and other methods. When PIC does better on a dataset, the corresponding point lies above the diagonal line.

Looking at k-means vs PIC accuracy chart, we see that there are clearly some "easy" datasets, with their corresponding points concentrated near the top right, and some "difficult" datasets concentrated near the bottom left. Aside from these, points lie mostly above the center diagonal line, showing that most of the time, PIC does as well or better

than k-means. There is not a strong correlation between k-means and PIC accuracy, possibly due to them being very different clustering methods, one using centroid-to-point similarity and one using all point-to-point similarities.

The $\text{NCut}_{\text{EVD}}$ vs PIC accuracy plot, with the exception of less than 10 datasets, forms a nearly diagonal line through the middle of the chart, showing that most datasets are "equally difficult" for these clustering methods. This may be an indication that the clusters produced by these methods are very similar, possibly due to them both using all point-to-point pair-wise similarity. We will not discuss $\text{NCut}_{\text{IRAM}}$ vs PIC accuracy here since $\text{NCut}_{\text{IRAM}}$ seems to have failed completely on this dataset to produce approximate eigenvectors.

### 6.4.2 Similarity between PIC and NCut Clusters

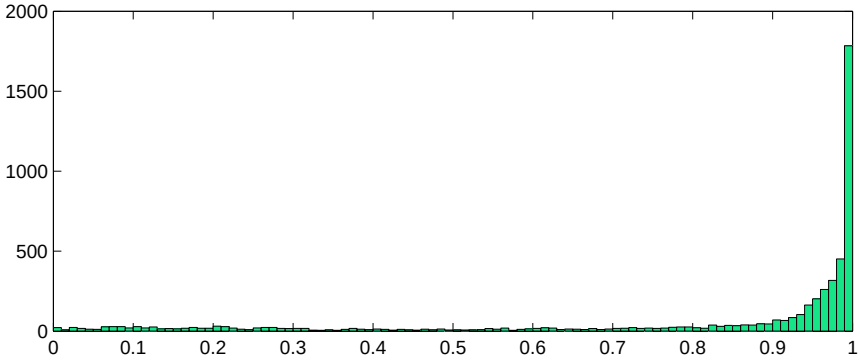



Figure 6.2: Histogram of the normalized mutual information between the clusters produced by PIC and by $\text{NCut}_{\text{EVD}}$. The x-axis corresponds to the NMI value and the y-axis correspond to the number of cluster comparisons.

Since PIC and NCut display a similar per-dataset accuracy performance as shown in Figure 6.1, and since that both methods derive their low-dimensional embedding from the normalized affinity matrix $W$, it makes sense to compare the output clusters of these two methods directly. Using the same 100 two-cluster datasets, we run both PIC and $\text{NCut}_{\text{EVD}}$ 50 times on each dataset. Each time a different $\boldsymbol{v}^0$ is used for PIC and a random set of initial k-means centers are used for both PIC and $\text{NCut}_{\text{EVD}}$, and the two

output clusters are paired. At the end we have 5,000 output pairs. We calculated the NMI between each of these pairs; the average NMI is 0.832 and a histogram is shown in Figure 6.2.

### 6.4.3 Scalability Results

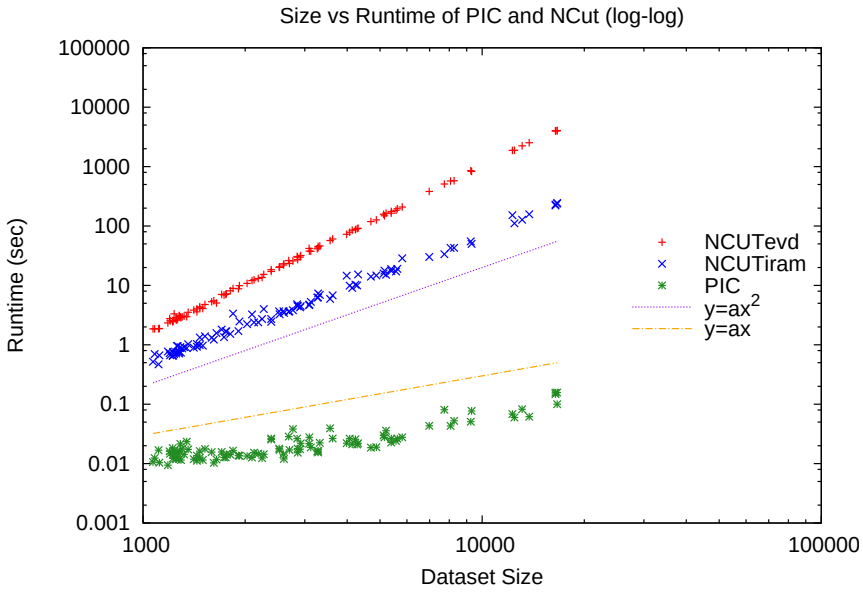



Figure 6.3: A size versus runtime plot on a log-log scale. The dots show runtime (in seconds) of various methods on datasets of increasing sizes. The lines show the slope of a linear curve and a quadratic curve for comparison purposes and do not correspond directly to any method.

We plot data size versus runtimes on a log-log chart in Figure 6.3. What is immediately noticeable is that PIC is much faster than either $\text{NCut}_{\text{EVD}}$ or $\text{NCut}_{\text{IRAM}}$. On the smallest dataset of 1,070 documents, PIC took only a hundredth of a second, 50 times faster than $\text{NCut}_{\text{IRAM}}$ and 175 times faster than $\text{NCut}_{\text{EVD}}$. On the largest dataset of 16,636 documents, PIC took about a tenth of a second, roughly 2,000 times faster than $\text{NCut}_{\text{IRAM}}$ and 30,000 times faster than $\text{NCut}_{\text{EVD}}$.

Note this time is with PIC calculating cosine similarities on-the-fly in each iteration, whereas NCut is given the pre-calculated cosine similarity matrix. What is even more remarkable is the asymptotic runtime behavior. To visualize this in the figure, we include

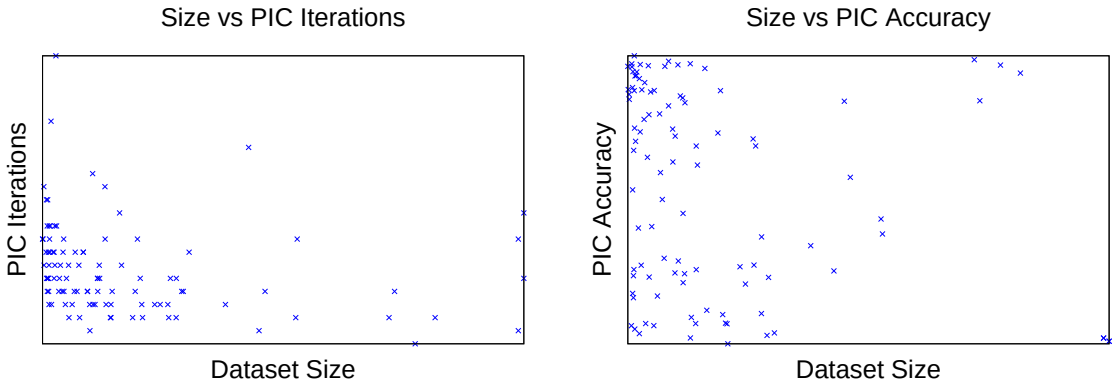
a line with quadratic behavior ($y = ax^2$) and a line with linear behavior ($y = ax$). With these are guidelines, we can see that $\text{NCut}_{\text{IRAM}}$ time is slightly above quadratic and $\text{NCut}_{\text{EVD}}$ close to cubic. PIC, on the other hand, displays a linear behavior. The runtime asymptotic behaviors of $\text{NCut}_{\text{EVD}}$ and $\text{NCut}_{\text{IRAM}}$ are more or less better understood [22] so these results are no surprise.

As Algorithm 2 and Equation 6.6 show, the runtime of each PIC iteration is strictly linear to the size of the input—that is, linear to the number of non-zero elements in the input document vectors. Assuming the vocabulary size is constant, then PIC runtime is:

$$O(n) \times (\#\text{ of PIC iterations})$$

Generally, it is difficult to analyze the number of steps required for convergence in an iterative algorithm (e.g., k-means), but if we are interested in the asymptotic behavior on certain datasets, we can instead ask a simpler question: does the number of iterations increase with dataset size? To observe this experimentally, we plot a correlation chart of the size of the dataset and the number of PIC iterations and calculate the $R^2$ correlation value, shown in Figure 6.4a. We find no noticeable correlation between the size of the dataset and the number of PIC iterations. This implies that the number of iterations is independent of dataset size, which means that asymptotically, the number of iterations is constant with respect to dataset size.

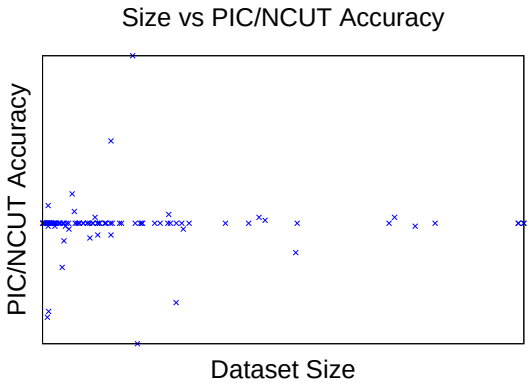
What if larger datasets are more "difficult" to PIC? It is less meaningful if the linear algorithm fails to work on bigger datasets. To observe this we calculate $R^2$ values and plot correlations between dataset size and PIC accuracy in Figure 6.4c and between dataset size and the ratio of PIC accuracy to $\text{NCut}_{\text{evd}}$ accuracy in Figure 6.4b. Again, with no discernible correlation in these figures, we conclude that PIC accuracy is independent of dataset size (Figure 6.4c) and that PIC is as accurate as NCut as dataset size increases (Figure 6.4b).



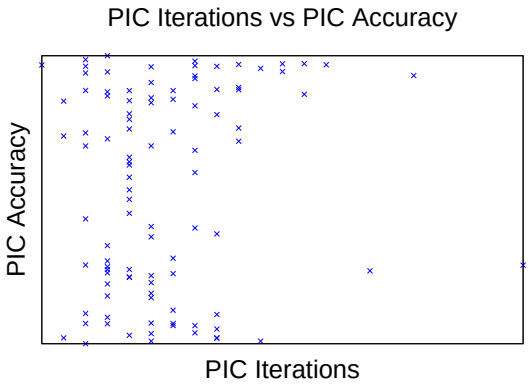

(a) $R^2 = 0.0424$




(b) $R^2 = 0.0552$




(c) $R^2 = 0.0007$




(d) $R^2 = 0.0134$

Figure 6.4: Correlation plots and $R^2$ correlation values. None of these plots or $R^2$ values indicates even a slight correlation, providing further evidence of PIC's runtime linearity. On average it takes 15 iterations for PIC to converge, with 31 iterations being the maximum.

An additional correlation statistic that may be of interest between that of PIC's accuracy and number of iterations. It is not unusual for an iterative algorithm to converge much faster on a "easy" dataset and slower on a more "difficult" dataset. Since the number of iterations is directly related to runtime, we may expect PIC to be slower on more "difficult" datasets. Figure 6.4d does not show correlation between the two, indicating that PIC work just as fast on "difficult" datasets as on "easy" datasets. With these results we conclude that, as far as text datasets are concerned, PIC's runtime is linear with respect to input size.

Perhaps PIC's runtime scalability is only matched by its small memory footprint. In addition to the input dataset, the "bipartite graph" PIC embedding requires *exactly* $4n$ storage ($v^t, v^{t-1}$ and diagonal matrix $\delta^{t-1}, D$) for inner product similarity and $5n$ (an additional diagonal matrix N) for cosine similarity, *regardless of vocabulary size*. This is much more feasible compared to at least $n^2$ storage required by methods requiring explicit construction of a similarity matrix.

## 6.5 Implicit Manifolds for SSL

Similar to PIC for clustering, the implicit manifold framework can also be applied to graph-based SSL methods where the core operations are iterative matrix-vector multiplications. MultiRankWalk (MRW) and the harmonic functions method (HF), introduced in Chapter 4 and described in more details in Chapter 5, are representative of two types of graph SSL methods that are a good "fit" for implicit manifolds. Algorithm 4 and 5 correspond to the basic MRW algorithm and an iterative solution implementation of HF, respectively.

**Algorithm 5** An iterative harmonic functions algorithm

1: **procedure** HARMONIC$_{IT}$($A, Y^L$)
2: $V^0_{ci} \leftarrow 1$ if $Y^L_i = c$, else $V_{ci} \leftarrow 0$ and $t \leftarrow 0$
3: **repeat**
4: $V^{t+1} \leftarrow D^{-1}SV^t$;
5: $\forall i \in Y^L : V^{t+1}_i \leftarrow V^0_i$ and $t \leftarrow t+1$
6: **until** $V^t$ has converged
7: $Y^U_i \leftarrow argmax_c(V_{ci})$
8: **return** $Y^U$
9: **end procedure**

Differences aside, an important algorithmic similarity between MultiRankWalk and Harmonic$_{IT}$ is that both have as their core operation iterative matrix-vector multiplications (Step 6 and Step 4, respectively). This allows us to replace the costly dense matrix

as in Equation 6.1, and the iterative operations for MRW and HF become:

$$\begin{aligned} \text{MRW:} \quad & V^{t+1} \leftarrow (1-\alpha)FF^T D^{-1} V^t + \alpha R \\ \text{HF:} \quad & V^{t+1} \leftarrow D^{-1} FF^T V^t \end{aligned}$$

Again, the usefulness of this representation is made more apparent if we specify the order of multiplication:

$$\begin{aligned} \text{MRW:} \quad & V^{t+1} \leftarrow (1-\alpha)F(F^T(D^{-1} V^t)) + \alpha R \\ \text{HF:} \quad & V^{t+1} \leftarrow D^{-1}(F(F^T V^t)) \end{aligned}$$

For document similarity, we can similarly apply Equation 6.3 to the matrix-vector multiplication step. Then the iterative operations for MRW and HF become:

$$\begin{aligned} \text{MRW:} \quad & V^{t+1} \leftarrow (1-\alpha)N(F(F^T(N(D^{-1} V^t)))) + \alpha R \\ \text{HF:} \quad & V^{t+1} \leftarrow D^{-1}(N(F(F^T(NV^t)))) \end{aligned}$$

For bipartite graph walk similarity, again we apply Equation 6.4 and 6.5 for MRW and HF:

$$\begin{aligned} \text{MRW:} \quad & V^{t+1} \leftarrow (1-\alpha)F(C^{-1}(F^T(R^{-1} V^t))) + \alpha R \\ \text{HF:} \quad & V^{t+1} \leftarrow R^{-1}(F(C^{-1}(F^T V^t))) \end{aligned} \tag{6.7}$$

### 6.5.1 Datasets

We carry out a series of experiments to see whether these graph-based SSL methods are effective on large, non-graph data under our implicit manifold framework and to see how HF and MRW compare against each other. For all experiments we use the

MATLAB implementation of SVM for a supervised learning baseline in a one-versus-all setting. We run HF fixed at 10 iterations (for reasons noted later), and we run MRW to convergence with parameter $\alpha = 0.25$.

We assemble a collection of four datasets; they are all from the text domain and are of two very different types of text datasets. The first type is *document categorization*, where the data is a collection of documents and the task is to predict category labels for each document. Here an instance is a document and the features are word occurrence counts. The second type is *noun phrase categorization*, where the data is a collection noun phrases (NPs) extracted from web and the context in which they appear. The task is to retrieve NPs that belong to the same category as a small set of "seed" NPs. For example, given "Seattle" and "Chicago" as seeds, we would like to retrieve NPs such as "Pittsburgh", "Beijing", and all other NPs corresponding to cities. Statistics of the datasets are found in Table 6.2, and note the memory requirement of using implicit manifolds (IM Size) versus constructing explicit manifolds (EM Size). We will describe each dataset in more detail in the following sections.

We choose implicit manifolds based on prior knowledge of what similarity functions work well on each type of data. For document collections we use cosine similarity [72]. For NP-context data, this particular task of NP categorization has been performed successfully using co-EM [81], which is closely related to the bipartite graph walk manifold.[3]

### 6.5.2 Document Categorization Results

The 20 Newsgroups dataset (20NG) is a collection of approximately 19K newsgroups documents, roughly 1,000 per newsgroup [75]. The class labels are the newsgroups groups; the features are word tokens, weighted according to the log-normalized TF-IDF scheme [62, 72]. The Reuters Corpus Volume 1 (RCV1) dataset is a benchmark

[3] In fact, the harmonic functions method with bipartite graph walk manifold is exactly equivalent to the co-EM algorithm used for performing information extraction from free text proposed by [50]. See Appendix C for details.

| Name | 20NG | RCV1 | City | 44Cat |
|---|---|---|---|---|
| Instances | 19K | 194K | 88K | 9,846K |
| Features | 61K | 47K | 99K | 8,622K |
| NZF | 2M | 11M | 21M | 121M |
| Cats | 20 | 103 | 1 | 44 |
| Type | doc | doc | NP | NP |
| Manifold | cosine | cosine | bipart | bipart |
| Input Size | 39MB | 198MB | 330MB | 2GB |
| IM Size | 40MB | 207MB | 335MB | 2.4GB |
| EM Size | 5.6GB | *540GB | *80GB | *4TB |

Table 6.2: Dataset comparison. *NZF* is the total number of non-zero feature values and *Cats* is the number of categories. *Type* is the dataset type, where *doc* and *NP* correspond to document collection and noun phrase-context data, respectively. *Manifold* is the choice of manifold for the dataset, where *cosine* and *bipart* refers to cosine similarity and bipartite graph walk, respectively. *Input Size* is the MATLAB memory requirement for the original sparse feature matrix; *IM Size* is the total memory requirement for using the implicit manifold, including the feature matrix; *EM Size* is the memory requirement for constructing a explicit manifold. An * indicates that the memory requirement is estimated by extrapolating from a random sample.

collection of 804K newswire stories [62]. We use the test split of 781K documents and *industries* category for labels. To simplify evaluation, documents with multiple labels and categories with less than 500 instances were removed, following previous work [22]. We ended up with 194K documents and 103 categories. We evaluate HF and MRW performance on these multi-class categorization datasets with the macro-averaged F1 score, where F1 score is the harmonic mean between precision and recall [72]. We randomly select a small number of instances per class as seeds and vary that number to observe the its effect on classification accuracy. The choice of manifold here is the cosine similarity commonly used for comparing document-document similarity. We also compare the results to that of SVM, a supervised learning method that has been proven to be state-of-the-art on text categorization datasets. The results for 20NG and RCV1 are shown in Figure 6.5

We see that SVM, the tried-and-true text classifier, outperforms both HF and MRW on these text categorization datasets, though MRW's performance is nearly as good as SVM on 20NG. HF does very poorly on both datasets. The difficulty of RCV1 may due to the imbalance in class distribution; the largest category consists of 23K documents where as the smallest consists of only 504 documents.

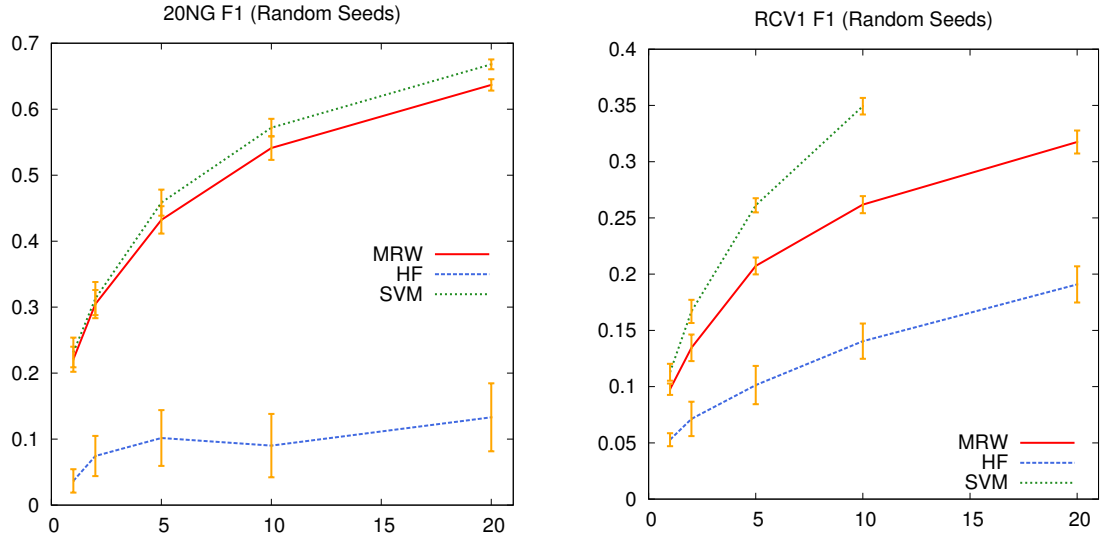



Figure 6.5: F1 scores on the 20NG and RCV1 datasets. The x-axis indicates the number of labeled instances and the y-axis indicates the macro-averaged F1 score. Vertical lines indicate standard deviation (over 20 trials for 20NG and 10 for RCV1) using randomly selected seed labels.

A notable result from previous work comparing HF and MRW on network data is that the "quality" of seeds (labeled training examples) is extremely important and makes a marked difference when only a few seeds are given [65]. We are interested to see if the same can be observed in document categorization. In network datasets good seeds can have a high degree (neighboring many nodes) or a high PageRank score (popular or authoritative); these nodes propagate labels better and are arguably much easier for a human to label, making them more cost-effective. Here we rank the quality of instances by its *feature-sum*—the sum of all of its feature weights. This roughly corresponds to the length of the document; it can be argued that longer documents have more information

to allow human labelers to confidently assign it to the right category, and they also make better seeds because they are able to propagate their labels through a larger set of features.

To verify this, we first rank all instances by their feature-sum and pick the top k instances from each category to be seeds; the results are shown in Figure 6.6. While HF does not seem to improve at all, MRW does improve on both datasets, with more dramatic improve on 20NG with a small number (1, 2, and 5) of seeds, outperforming SVM. An interesting note is that SVM performance suffered on 20NG with high feature-sum seeds; a probable explanation is that high feature-sum seeds are likely to be "central" instances within its category cluster in the feature space, and whereas central instances are good for propagating their labels within its cluster, they are not good "boundary" instances that make good *support vectors* as required by margin-based classifiers such as SVM.

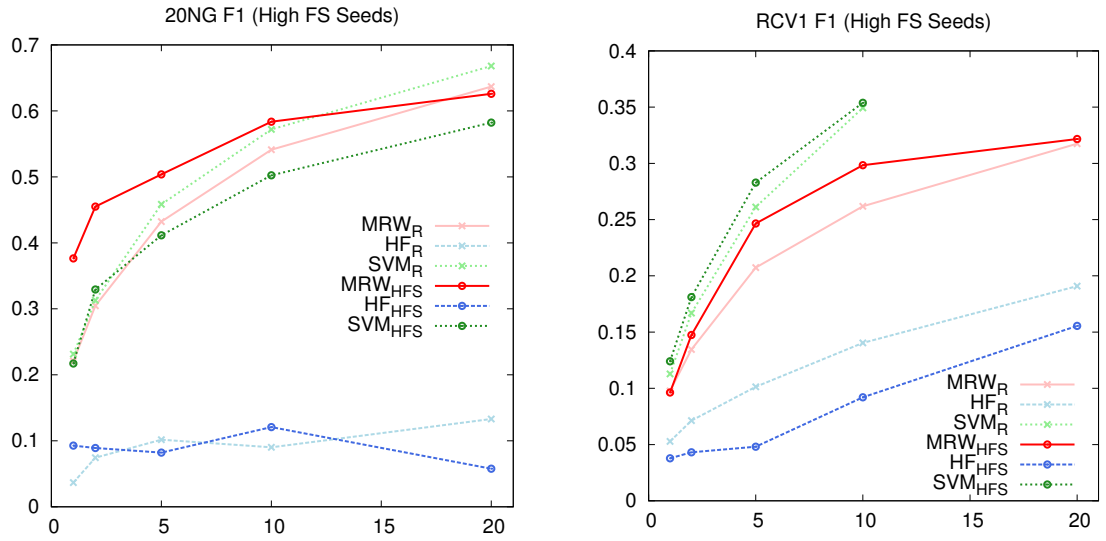



Figure 6.6: F1 scores on the 20NG and RCV1 datasets using preferred (high feature weight sum) seeds. Subscript HFS indicates result using high feature-sum seeds and R indicates result using random seeds—included for comparison.

### 6.5.3 Noun Phrase Categorization Results

The City and 44Cat datasets are derived from the NP-context data described in [18]. The NP-context data is extracted from a collection of approximately one billion web pages; unique English sentences are extracted from these web pages and a noun phrase chunker is ran over these sentences to extract a set of noun phrases (NPs) and their surrounding word patterns (contexts). Statistics of the co-occurrence counts of these NPs and contexts makes up a the NP-context dataset. Here we use this data for NP categorization; in this framework, each unique NP is an instance and its co-occurring contexts are features. For example, if the NP "Pizza" is found to co-occur with the context "we ordered _" 2 times and "I love _" 5 times, then "Pizza" would have these features with weights 2 and 5, respectively. The choice of graph-based SSL manifold for this dataset is the bipartite graph walk manifold because HF with this manifold is closely connected to the co-EM algorithm (see Appendix C), which worked well on this type of data [50]. Our general framework enables us to apply the same manifold to MRW as well.

City is the smaller dataset which consists of the most common (occurrence $> 500$) 88K NPs and 99K contexts, 7 "city" and 14 "non-city" hand-picked seeds. We also have ground truth labels for all of the NPs, created thus: first we obtained an exhaustive list of city names from World Gazetteer [4]; by matching the NPs in the dataset with the list from World Gazetteer, we end up with $5,921$ NPs that are candidates belonging to the "city" category. However, many of these are obscure city names that never appear in the data *as cities*. To filter these false positives we use Amazon Mechanical Turk [1] and have human labelers decide whether a NP refers to a city according to its top 15 most frequent contexts. This resulted in a final list of $2,404$ "city" NPs.

Unlike document categorization datasets where every document has a label, here we are retrieving a small set of positive instances from a much larger set of uncategorized negative instances. Additionally, since the output of task (a list of NPs belonging to specified categories) has been used to create a high-quality ontology [19], we also want to see

| Method<br>Manifold | SVM<br>- | HF<br>inner | MRW<br>inner | HF<br>bipart | MRW<br>bipart |
|---|---|---|---|---|---|
| NDCG | 0.026 | 0.040 | **0.041** | **0.041** | **0.041** |
| AP | 0.021 | 0.673 | 0.707 | 0.713 | **0.739** |
| P@10% | 0.012 | 0.873 | 0.893 | 0.880 | **0.909** |
| P@20% | 0.014 | 0.870 | 0.900 | 0.895 | **0.916** |
| P@30% | 0.017 | 0.877 | 0.910 | 0.904 | **0.912** |
| P@40% | 0.020 | 0.857 | 0.896 | 0.912 | **0.918** |
| P@50% | 0.021 | 0.823 | 0.865 | 0.883 | **0.904** |
| P@60% | 0.024 | 0.760 | 0.799 | 0.809 | **0.831** |
| P@70% | 0.027 | 0.634 | 0.674 | 0.681 | **0.719** |
| P@80% | 0.027 | 0.413 | 0.453 | 0.509 | **0.530** |
| P@90% | 0.027 | 0.193 | 0.216 | 0.252 | **0.293** |
| P@100% | 0.027 | 0.028 | 0.028 | 0.028 | **0.029** |

Table 6.3: City dataset result. Boldface font indicates the highest number in a row; *inner* refers to the inner product manifold and *bipart* refers to the bipartite graph walk manifold.

if a classifier is able to assign higher confidence to correctly labeled NPs (i.e., we want the classifier to rank these NPs in order of their likelihood of being in a category). So for evaluating this dataset we choose to use measures for ranking quality from information retrieval literature: NDCG (normalized discounted cumulative gain), AP (average precision), and precisions at increasing level of recall.

Here for HF and MRW the confidence score is simply the ratio between the positive (city) score and the negative (non-city) score. For these experiments, we also add a smoothing parameter $\beta$ to MRW so that step 6 in Algorithm 4 becomes:

$$V^{t+1} \leftarrow (1-\alpha-\beta)AD^{-1}V^t + \alpha R + \beta(\mathbf{1}/n)$$

where $\alpha+\beta \leq 1$. Typically $\beta$ is very small, and can be considered an uniform smoothing factor on the confidence. Note that the label prediction does not depend on $\beta$ at all; this only affects the ranking to avoid problems such as divide-by-zeros and over-confidence

on ratios with very small values. The confidence ranking of SVM is determined by the distance of an NP to the margin of its assigned class.

The result is shown in Table 6.3: using the bipartite graph walk gives a noticeable advantage over inner product manifold, and MRW outperforms other methods. Here SVM does poorly due to feature sparsity, which highlights the effectiveness of graph-based SSL methods with a small number of seeds. Table 6.3 also illustrates an advantage of viewing co-EM as HF with a bipartite graph walk manifold—it shows that using this particular manifold improves performance for MRW as well as for HF.

| Top k | Sample | SVM | HF | MRW |
|---|---|---|---|---|
| 100 | 50% | 0.31 | 0.52 | 0.52 |
| 500 | 10% | 0.29 | 0.50 | 0.47 |
| 1000 | 5% | 0.29 | 0.48 | 0.47 |

Table 6.4: Averaged estimated precisions of the top 100, 500, and 1000 retrieved NPs on the 44Cat dataset. Precisions are averaged over 44 categories and sampled at a rate of 50%, 10%, and 5%, respectively. No statistical significance is found between the results of HF and MRW.

44Cat is a much larger NP-context dataset, consisting roughly 10 million English NPs found in one billion web pages and 9 million co-occurring contexts. We removed any NP-context with co-occurrence count less than three and used roughly 15 hand-picked NPs as seeds for each of the 44 categories as found in [19]. We do not have a set a ground truth labels for these categories prior to running experiments, as obtaining them would be extremely expensive. Instead, we first obtained a list of 1,000 NPs for each category using SVM, HF, and MRW, ranked by their confidence. Then we computed the *estimated precision* for the top 100, 500, and 1,000 NPs of each ranked list, estimated by judging the precision at a 50%, 10%, and 5% sample of the NPs, respectively. The judging is again done using AMT; every sampled prediction is given three workers, and when they disagree we take the decision of the majority. The algorithm settings for this dataset is the same as the City dataset. An overall result, averaged across 44 categories, is shown in Figure 6.4. Here we see that again SVM does poorly compared to graph-based SSL

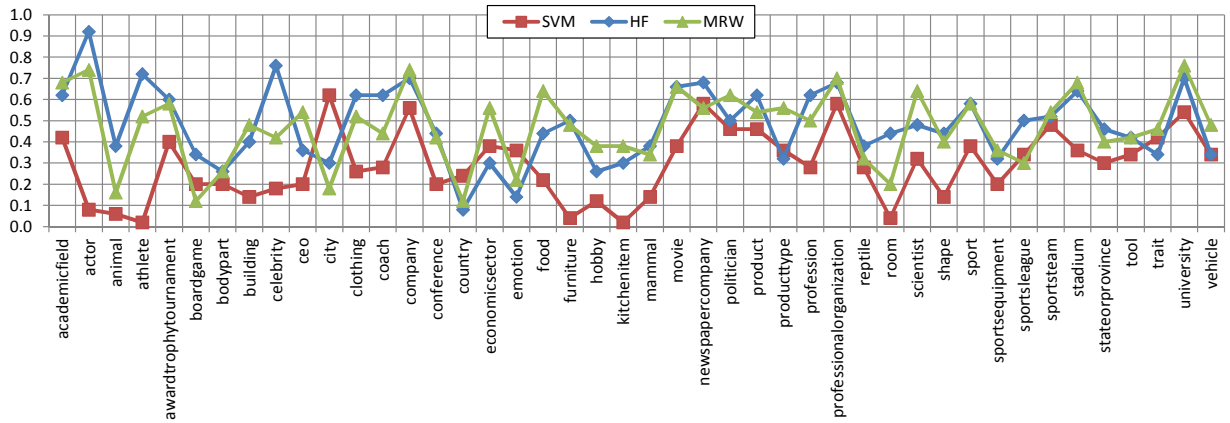



Figure 6.7: Sampled per-category accuracies of the top 1000 retrieved NPs on the 44Cat dataset. The categories are ordered from left to right according to the difference between the MRW accuracy and HF accuracy, from the high to low.

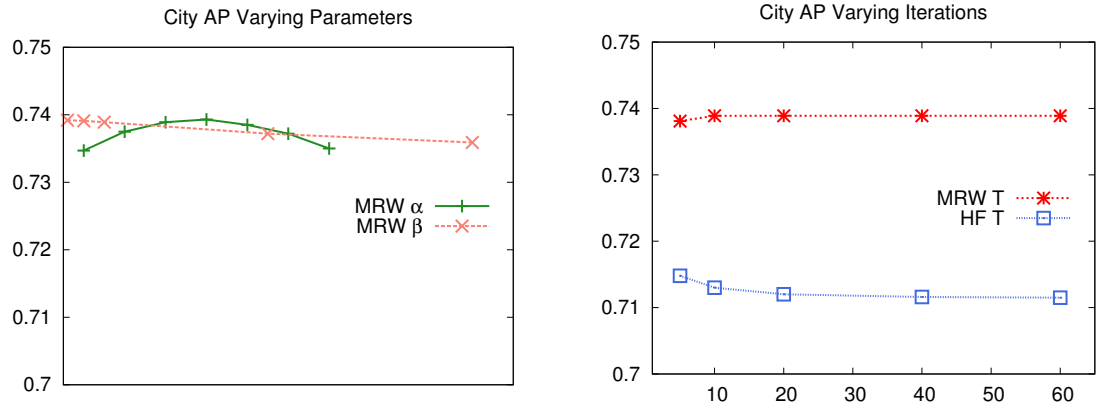
methods. Both HF and MRW are equally effective on this dataset with no statistically significant difference. We conjecture that the lack of statistical differences may be due to the relatively small number of samples per category and large per-category differences; for example, for the top 1,000 NPs, HF did the best on 23 categories and MRW on 22 (with ties in 4 categories).

We also break down the results of the estimated precision at top 1,000 retrieved NPs into 44 categories in Figure 6.7. In this figure the categories are arranged from left to right in order of the difference in precision between MRW and HF. An immediate observation is that no one method can claim to be the best on all categories, though we see that for most categories SSL methods outperforms SVM by a good margin. Note that points where all three lines dip on the chart correspond to categories that have a small, closed set of items (e.g., "country" and "bodypart"), this is understandable since, for example, the number of countries in the world is much less than 1,000. Results for the top 100 and 500 NPs can be found in Appendix B.3.

### 6.5.4 Runtime Efficiency

Implicit manifolds also yield fast runtimes. The smallest dataset (20NG) takes less than 1 second for both HF and MRW, and the largest (44Cat) takes less than 3 minutes. We did not try running explicit manifold versions of the algorithm for runtime comparison because that would require more memory than we have available for most of these datasets (See Table 6.2). Experiments were done on a Xeon 2.27GHz Linux machine using a MATLAB implementation.

### 6.5.5 Parameter Sensitivity




(a) $\alpha$ and $\beta$ (b) the # of iterations $\tau$

Figure 6.8: Parameter sensitivity. The x-axis correspond to parameter values and the y-axis shows average precisions; $\alpha$ ranges from 0.05 to 0.65, $\beta$ ranges from 0.0001 to 0.01; the number of iterations $\tau$ is indicated below the x-axes.

The parameter sensitivity of a method is an issue for larger datasets where tuning or sweeping of parameters may be impractical. Parameters for MRW are $\alpha$, $\beta$, and the number of iterations $\tau$ (correlated with the convergence threshold). The only parameter for HF is $\tau$. We plot varying parameters and their effect on average precision in Figure 6.8. Here neither method appears to be very sensitive. Note that HF's AP peaks around $\tau = 5$ and it actually *degrades* with more iteration. This suggests early-stopping HF may

yield better performance—hence why we fixed $\tau = 10$ for HF. This also points to an advantage MRW has over HF—unlike HF, MRW does not seem to "over-iterate".

### 6.5.6 Mislabeling Resistance

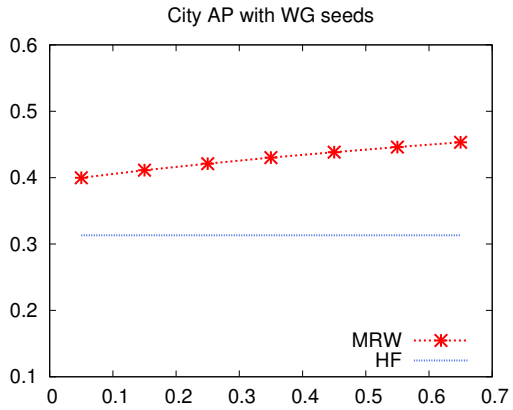



Figure 6.9: City dataset result using all the World Gazetteer-matched NPs as seeds. The x-axis correspond to varying $\alpha$ values and the y-axis indicates the average precision.

An important difference between HF and MRW is that for HF, all seeds are "equal" in the sense that every seed node has the same classification influence on its neighbors, whereas in MRW, a more "central" seed node (one closer to other seed nodes) has more influence on its neighbors. For example, a node close to a central seed node is more likely to be positive than a node close a non-central seed node. This property of MRW becomes useful when the seeds are noisy (e.g., some are mistakenly labeled positive). To illustrate this, we give both HF and MRW as input seeds the entire set of 5,921 NPs that matched city names in World Gazetteer, and evaluate the outputs as before. The result is shown in Figure 6.9. As expected both methods are less accurate than those using hand-picked seeds, but MRW is able to do significantly better. This is because noisy seeds are more likely to be isolated—i.e., "less central". Note also that unlike Figure 6.8 the average precision of MRW is slightly higher with a higher value of $\alpha$—this is perhaps due to all city NPs are already seeds (propagating to non-seed nodes do not help), and a

higher restart probability concentrate the walk probability around more "central" seeds, distinguishing them from the noisy seeds.

## 6.6 Discussion

*Manifold learning* [48] is an area of study in machine learning and data mining where the tasks are often to construct (or learn) an appropriate similarity function or a nonlinear dimensionality reduction mapping of the data for subsequent clustering, classification, or visualization. In this regard, implicit manifolds is related to manifold learning in that it provides efficient solutions for a class of pair-wise similarity manifolds in combination with a class of learning algorithms where an explicit construction of the similarity matrix would be impractical. However, though similarly named, the general idea behind implicit manifolds can be applied to a much wider array of computational methods beyond manifold learning—as long as the method has at its core matrix-vector multiplications and the matrix is composed of, or can be decomposed into, sparse matrices.

# Chapter 7

# Edge Clustering

## 7.1 Introduction

As discussed in Section 2.2 and 3.1, *Spectral clustering* [103] is a data clustering paradigm where the bottom eigenvectors of a specific Laplacian (e.g., the Normalized Cuts Laplacian [91] or the symmetric normalized Laplacian [80]) of the affinity matrix of the data points are used to construct a low-dimensional embedding in which clusters are clearly separated in a metric space. Spectral clustering is popular due to its simplicity, effectiveness, and its ability to deal with non-linearly separable clusters.

There are two major drawbacks to spectral clustering methods. The first drawback is that they are computationally expensive because of the eigenvector computation, which is non-trivial even with faster sparse and approximate techniques [22]. As discussed and shown in Chapter 2, *power iteration clustering* (PIC) ameliorates this drawback, providing a highly scalable approach that outputs a similar clustering. The second drawback is that, as a type of *graph partition* method, unlike probabilistic topic models [17] or probabilistic network models [6, 11], spectral methods do not allow mixed membership clustering (overlapping clusters).

To address these problems, first we describe a method for converting a node clustering (graph partition) method into a mixed membership clustering approach by transforming a node-centric representation to a scalable edge-centric representation, and then we describe how this transformation can be used with PIC. We show a substantial improvement in performance in mixed membership clustering tasks using the proposed approach.

## 7.2 Edges, Relationships, and Features

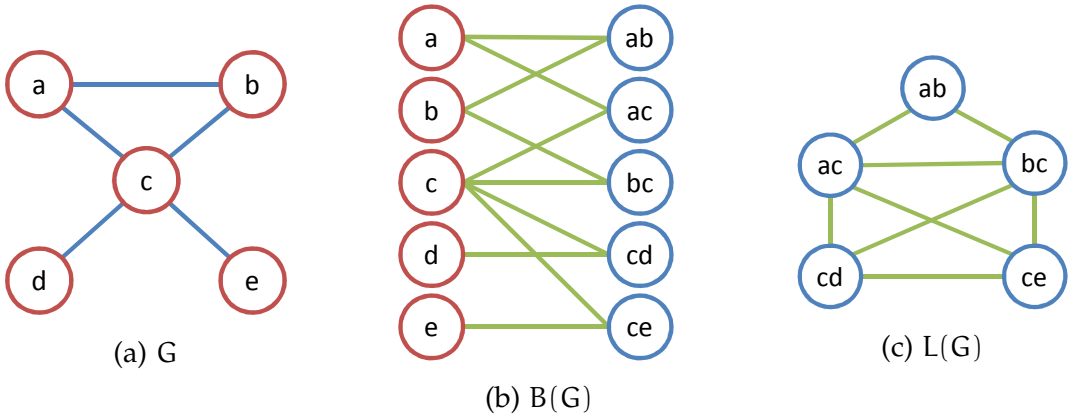



Figure 7.1: An example graph G and its corresponding bipartite feature graph B(G) and line graph L(G). In (b) and (c), the blue nodes represent the edges in G; e.g., edge ab represents the edge connecting a and b in (a).

To achieve a mixed membership effect, instead of clustering the nodes in the graph, we propose to cluster the edges in the graph.

A central assumption we make in this work is that, while nodes in a graph can belong to more than one cluster, an edge between two nodes, indicating an affinity relationship, belongs only to one cluster. If we can determine the membership of these edges correctly, then we can assign multiple labels to the nodes based on the membership of the incident edges. In the context of a social network, edge clustering can be interpreted as relationship clustering—instead of forcing every person to belong to only one social community,

each of the relationships will be assigned to a social community. For example, person $a$'s relationships with $a$'s parents, siblings, and cousins belong to the community of $a$'s family and relatives, and $a$'s relationships with his co-workers belong to the community of the company $a$ works for. One-community-per-relationship is a much better assumption than one-community-per-person because it better fits our understanding of a social network structure and allows multiple community labels per person—in fact, person $a$ can have as many labels as the number of relationships $a$ has.

If we are to apply graph partition methods to edges of the graph, first we need to transform the graph so to an "edge-centric" representation. In this work, we will construct what we call a *bipartite feature graph* (BFG). A BFG $B(G)$ on a graph $G$ is a bipartite graph satisfying the following conditions:

- $B(G) = (V_V, V_E, E)$ where $V_V$ and $V_E$ are disjoint sets of nodes and $E$ is the set of edges.
- Each node in $V_V$ corresponds to a node in $G$ and each node in $V_E$ corresponds to an edge in $G$.
- An edge $e \in E$ exists between nodes $a \in V_V$ and $b \in V_E$ if and only if node $a$ is incident to edge $b$ in $G$.

In other words, for each edge $e_i(u, v)$ in $G$, we add a new node $e_i$ to the node set of $B(G)$, and connect $e_i$ to the nodes $u$ and $v$ in $B(G)$. See Figure 7.1 for an example.

We compare BFG to *line graphs*, a more common edge-centric representation. A line graph $L(G)$ on a undirected, unweighted graph $G$ is a graph where (1) each node of $L(G)$ represents an edge of $G$, and (2) an edge exists between two nodes of $L(G)$ if and only if their corresponding edges in $G$ have an incident node in common; In other words, for each edge $e_i(u, v)$ in $G$, we create a node $e_i$ in $B(G)$, and connect $e_i$ to $e_j$ if $u$ or $v$ is also an end point of $e_j$.

There are several advantages to BFG as an edge-centric representation over line graphs: (a) unlike a line graph, the original graph $G$ can always be constructed from

a BFG $B(G)$, (b) $B(G)$ has the same space complexity as that of $G$, and (c) it is trivial to modify the BFG to correspond to a directed, weighted graph. The biggest drawback with line graphs is that they not scalable. For example, a set of nodes and edges in $G$ that form a "star" pattern—one node in the middle connected to $n$ nodes via $n$ edges—translates to $n$ nodes and $n^2$ edges in $L(G)$. This is especially a problem in most large, social networks that display a power-law distribution in the number of incident edges per node [33,61].

A graph $G$ can be represented as a square matrix $A$ where the rows and columns correspond to nodes and a non-zero element $A(i,j)$ corresponds to an edge between nodes $i$ and $j$; similarly, a BFG $B(G) = (V_V, V_E, E)$ can be represented as a rectangular matrix $F$ where the rows correspond to $V_E$ and the columns correspond to $V_V$, and a non-zero element $F(i,j)$ correspond to an edge between $i$ and $j$, which in turn represents an incidence of edge $i$ on node $j$ in $G$. An immediate corollary is that $F$ will always be sparse, since every row in $F$ will always have only two non-zero elements, and the number of non-zero elements in $F$ is $2m$ where $m$ is the number of edges in $G$. Thus, while a BFG transformation does not increase the space complexity of the original input, methods that work with the BFG need to be sparse matrix-friendly in order to scale to large datasets.

An important observation we want to make here is that a $B(G)$ is a valid edge-centric representation of $G$ regardless of $G$'s structure, even if, let's say, $G$ is itself a bipartite graph. This observation generalizes BFG to represent not just graphs, but any data represented by weighted feature vectors. Actually, a bipartite graph has been often used to represent large feature vector-based datasets such as noun phrases found on the web [67,93], large document collections [66,67], and social network communities [94,104] for scalable semi-supervised learning and clustering analysis.

A bipartite graph can be constructed from a dataset of feature vectors $X$ as follows. Let $\mathbf{x}_i \in X$ be the $i$-th instance of $X$, and let $\mathbf{x}_i(j)$ be the weight of $\mathbf{x}_i$'s $j$-th feature. Then

create a bipartite graph $G = (V_I, V_F, E)$ where the i-th node in $V_I$ correspond to the $\mathbf{x}_i$ and the j-th node in $V_F$ correspond to the j-th feature, and an edge $e(i,j) \in E$ is weighted by $\mathbf{x}_i(j)$. The BFG can be applied as above and therefore we can transform any graph or dataset of feature vectors into an edge-centric representation.

An intuitive interpretation of edge clustering on the bipartite instance feature graph is that, instead of clustering instances, where each instance can belong to multiple clusters, we want to cluster each *feature occurrence*. For example, a text document on the subject of sports management may belong to both "sports" and "business administration" categories. However, we can assign each word occurrence to a specific category (e.g., the word "football" to the sports category and the word "budget" to the business category), and then assign the document multiple labels depending on the labels of its word features.

## 7.3 Edge Clustering

After transforming a graph into a BFG, edge clustering can be done with any graph-based clustering method, like spectral clustering methods such as Normalized Cuts [91] and the Ng-Jordan-Weiss method [80]. Then the cluster labels assigned to the edge nodes $V_E$ in the BFG can be used to determine mixed membership labels for the nodes in the original graph. Algorithm 6 outlines the steps for the general procedure, where $A$ is the matrix representation of the input graph, $k$ is the number of desired clusters, and Cluster and Labeler are the specified clustering method and the node label assignment strategy, respectively.

An issue to consider when choosing the graph clustering method is its scalability. If the original graph $G = (V, E)$ is represented by a $|V| \times |V|$ matrix, then its BFG matrix is $(|E| + |V|) \times (|E| + |V|)$, a much larger matrix for most types of data. In order for this

**Algorithm 6** General mixed membership clustering method via edge clustering

1: **procedure** MMCluster($A$, k,Cluster,Labeler)
2: Transform $A$ into a BFG and get $B(A)$
3: Run Cluster($B(A)$, k) and get edge clusters $E_1, E_2, ..., E_k$
4: Run Labeler($E_1, E_2, ..., E_k$) and get mixed membership node clusters $C_1, C_2, ..., C_k$
5: **return** $C_1, C_2, ..., C_k$
6: **end procedure**

approach to scale to large datasets, the graph-based method must be able to take full advantage of the sparsity of BFG.

### 7.3.1 Edge Clustering with PIC

For the graph-based clustering method we use PIC, discussed in detailed in Chapter 2. Here we rewrite the PIC algorithm, Algorithm 2 in Chapter 2, as Algorithm 7. Here we rewrite $W$ as $D^{-1}A$ and separate Steps 4 and 5 so it is easier to explain how to adapt it for edge clustering.

**Algorithm 7** The PIC algorithm

1: **procedure** PIC($A$, k)
2: Initialize $\boldsymbol{v}^0$ and $t \leftarrow 0$
3: **repeat**
4: $\boldsymbol{v}^{t+1} \leftarrow D^{-1}A\boldsymbol{v}^t$
5: Normalize $\boldsymbol{v}^{t+1}$
6: $\delta^{t+1} \leftarrow |\boldsymbol{v}^{t+1} - \boldsymbol{v}^t|$ and $t \leftarrow t+1$
7: **until** $|\delta^t - \delta^{t-1}| \simeq 0$
8: Use k-means on $\boldsymbol{v}^t$ to get clusters $C_1, C_2, ..., C_k$.
9: **return** $C_1, C_2, ..., C_k$
10: **end procedure**

The diagonal degree matrix $D$ is defined as $D(i,i) = \sum_j A(i,j)$. Typically, the indicator $\boldsymbol{v}^0$ is first assigned uniformly random values, as in the power method for determining the principle component of a square matrix. The normalization in Step 5 can be used to keep the numerical values in $\boldsymbol{v}$ from overflow or underflow, and can also used to keep $\boldsymbol{v}$ a probability distribution (e.g., $\boldsymbol{v}^{t+1} \leftarrow \frac{\boldsymbol{v}^{t+1}}{\|\boldsymbol{v}^{t+1}\|_1}$).

The input to PIC is $A$, a non-negative square affinity matrix where $A(i,j)$ represents the similarity between instances $i$ and $j$ (or, in the context of a graph, the weight of the edge between node $i$ and $j$). If we want to cluster the nodes of a graph $G$ (assuming homophily), then we can simply use $G$ for $A$. However, here we want to cluster the edges of $G$. As mentioned in Section 7.2, the most direct edge-centric representation of $G$ is the line graph $L(G)$, but the problem with $L(G)$ is that not only is it potentially a very dense graph, it will most likely be dense for many types of data such as social networks and document collections. The bipartite feature graph $B(G)$ is a more scalable representation, but it introduces another problem. A BFG, which is a bipartite graph, is always *periodic*, and therefore iterative algorithms such as the power iteration and PageRank [82] do not converge. Likewise, PIC, which is based on the power iteration, cannot be used on $B(G)$.

Here we propose to turn $B(G)$ into a unipartite graph as follows. Let $cn(i,j) \subseteq V_V$ be the set of common incident nodes of edge $i$ and $j$ in $G$, and let $F$ be the matrix representation of $B(G)$ (as in Section 7.2), and we define a similarity function $s(i,j)$ where $i,j \in V_E$ as follows:

$$s(i,j) = \sum_h \frac{1}{\sum_j F(j,h)} F(i,h) \cdot F(j,h) \tag{7.1}$$

In other words, the similarity between edge $i$ and $j$ in $G$ is the number of common incident nodes they have (at most 2), weighted proportionally to the product of the edge weights but inversely proportional to the number of edges each node is incident to. This is an intuitive similarity function between two edges that incorporates the number of common incident nodes, their weights, and the inverse frequency of the incident nodes[1]. Then we can define a square matrix $S$ where $S(i,j) = s(i,j)$, and use $S$ in place of $A$ in Algorithm 7.

[1]The assumption is that a node should not be considered important for determining similarity if it is incident to many edges; e.g., two links both pointing to the popular search engine Google.com is hardly a evidence of their similarity. This is similar to the *tf-idf* term weighting commonly used for comparing document similarity in information retrieval methods [72]

One final difficulty remains: $S$ could still be dense. As Equation 7.1 implies, $s(i,j)$ is non-zero as long as $i$ and $j$ have one incident node in common—which brings us back to the problem with line graphs mentioned in Section 7.2. However, the *implicit manifold* "trick" in Chapter 6 can be applied to obtain a solution with time and space complexity linear to the size of the input because (a) The (likely) dense similarity matrix $S$ can be decomposed as $S = FNF^T$, a product of sparse matrices, where the diagonal matrix $N$ defined as $N(j,j) = \sum_i F(i,j)$, and (b) $A$ is only used in Step 4 of Algorithm 7 for a matrix-vector multiplication. We can then replace Step 4 of Algorithm 7 with

$$\boldsymbol{\nu}^{t+1} \leftarrow D^{-1}(F(N(F^T\boldsymbol{\nu}^t))) \tag{7.2}$$

and produce the exact same result as using $S$ directly. As before, $D$ can computed efficiently by computing a vector $\mathbf{d} = FNF^T\mathbf{1}$ (where $\mathbf{1}$ is a vector of 1's) and let $D(i,i) = \mathbf{d}(i)$.

### 7.3.2 Assigning Node Labels

After obtaining a cluster label for every edge, we can proceed to assign labels for every node based on the labels for its incident edges. Let $L(i,j)$ be the number of node $i$'s incident edges assigned the $j$-th label. We propose three simple variations:

**Max** The label assigned to node $i$ is $\mathrm{argmax}_j L(i,j)$. This will assign only one label to a node as in typical node-based clustering methods, and is useful for comparing against them in a single-membership setting.

**T@p** Label $j$ is assigned to node $i$ if $\frac{L(i,j)}{\sum_j L(i,j)} \geq \frac{p}{100}$; i.e., T@20 means that node $i$ will be assigned label $j$ if at least 20% of its incident edges are assigned $j$. It falls back to Max if no labels meet the criteria.

**All** Label $j$ is assigned to node $i$ if $L(i,j) \geq 1$; i.e., node $i$ will be assigned all the labels of its incident edges.

Putting together the various parts of the method, the mixed membership clustering via edge-clustering using PIC, which call $\text{PIC}_\text{E}$, is outlined in Algorithm 8. Note that unlike Algorithm 6, $\text{PIC}_\text{E}$ uses the compact representation $F$ instead of $B(A)$.

**Algorithm 8** Mixed membership clustering using PIC

1: **procedure** $\text{PIC}_\text{E}(A, k, p)$
2: Transform $A$ into a BFG as $F$
3: Run PIC($F, k$), replacing Step 4 with Equation 7.2, and get edge clusters $E_1, E_2, ..., E_k$.
4: Use T@p on $E_1, E_2, ..., E_k$ to get node clusters $C_1, C_2, ..., C_k$.
5: **return** $C_1, C_2, ..., C_k$
6: **end procedure**

## 7.4 Experiments

For experiments we compare the proposed method with different node label assignment methods on a number of datasets. In addition we will use the original node-centric PIC and the closely related normalized cuts method (NCut) [91] as baselines. Instead of evaluating clustering methods indirectly using a supervised learning task as done in some previous work [94, 104], we want to compare output clusters directly with human assigned categories. For the evaluation metric we will report the macro-averaged F1 score[2], often used for multi-label categorization tasks [62, 72], after aligning output clusters with ground-truth category labels using the Hungarian algorithm. We prefer this metric over label accuracy because the latter tend to inflate prediction performance when the cluster sizes are not balanced.

### 7.4.1 Modified Network Datasets

We gather a set of eight benchmark network datasets with single-membership ground-truth labels for our first set of experiments. For each dataset we synthesize mixed membership instances by randomly drawing and merging pairs of nodes. We prefer mod-

[2]The harmonic mean of precision and recall

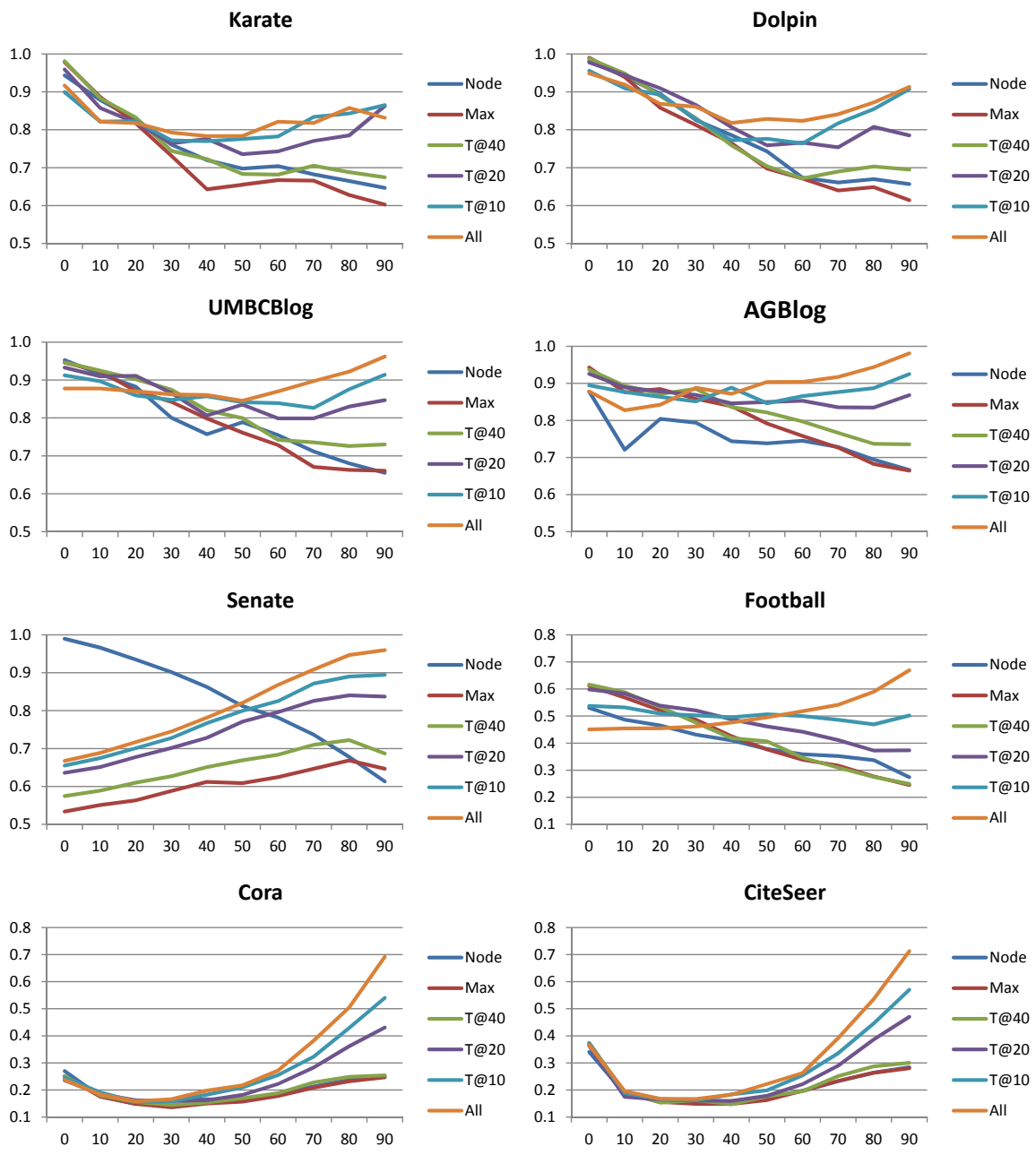



Figure 7.2: Mixed membership parameter m (x-axis) versus macro-averaged F1 score (y-axis) on the modified network datasets.

ifying a variety of existing real datasets over purely synthetic ones such as the *planted partition model* [24] as they may be a better predictor on how well these methods would perform on these types of real mixed membership datasets. Our primary goals for these experiments are:

- To verify that edge clustering works as well as node clustering on single-membership data.
- To vary the degree of "mixed membership-ness" and see how well each method does.

The merging process is follows. (1) Split nodes of graph $G = (V, E)$ into two sets $S$ and $T$ such that $S \cup T = V$ and $S \cap T = \emptyset$, such that $\frac{|S| \cdot 100}{|V|} \approx m$. (2) For each node $s \in S$, randomly select a node in $t \in T$, and add all labels and edges of $s$ to $t$ (i.e., add a new edge $(t, c)$ if $(s, c) \in E$). (3) Remove all nodes in $S$ and their incident edges.

The parameter $m$ controls the degree of mixed membership-ness and goes from 0 to 100, where 0 would result in the original single membership graph and 100 would result in a graph with a single node with all possible membership labels. In between, $m$ guarantees that at least $m\%$ of the nodes in original single membership graph $G$ is merged in the mixed membership graph $G'$.

For the experiments this process is repeated 50 times per dataset and the reported evaluations are averaged over these 50 runs.

Here we briefly describe the eight datasets. The Karate dataset [108] form a two-community social network among 28 members a karate club; the nodes are people and the edges are friendships. The Dolphin dataset [70] is a two-community social network of associations between dolphins in a pod in New Zealand. The UMBCBlog [51] and AGBlog [5], datasets are networks of 404 and 1222 political blogs, respectively. The nodes are blogs and edges are hyperlinks between them, and each blog belong either to the *liberal* or *conservative* community. The Senate dataset contains nodes corresponding to 98 US senators and edges are agreement on congressional votes; labels correspond to affiliations with either a *liberal* or a *conservative* political party; unlike other datasets, this dataset is a complete graph. The nodes in the Football dataset [41] are 115 US Division IA colleges, each belonging to one of 10 conferences, and the edges represent games in the 2000 regular season. The Cora and CiteSeer datasets [69] are 2485 and 2114 scientific

papers belonging to 7 and 6 related scientific fields, respectively; edges are citations. All of these datasets have weighted, undirected edges.

The experiment results on the network datasets are shown in Figure 7.2, where Node refers to the original PIC algorithm using Node-based clustering. Results for NCut is nearly the same as that of Node and is not shown in the figure for sake of clarity. Here we make a few observations: (a) Except for Senate, on most datasets, edge clustering methods do just as well as Node for $m = 0$ (single membership). (b) As $m$ increases, All and methods with a low $p$ perform better as expected. (c) The performance of Max is very similar to that of Node—it does well at low $m$'s but not at higher $m$'s, whereas All usually is the worst (not by much) at low $m$'s and best at high $m$'s. (d) The poor performance of edge clustering methods on Senate suggests that they may not be well-suited for certain dense network datasets. (e) The threshold parameter $p$ should be tuned for an optimal result—Max and All do not do well at particular extremes of $m$, while T@20 consistently outperform Node at almost any $m$, except on the Senate dataset. The results verify that edge clustering will generally work well on single-membership datasets as well as mixed membership ones. Note that at the very high end of $p$ edge clustering methods, especially All, would do well simply because most instances will have membership in most classes.

### 7.4.2 BlogCatalog Datasets

BlogCat1 and BlogCat2 [94, 104] are two blog datasets crawled from BlogCatalog.com, during two different time periods. BlogCat1 contains 10,312 blogs/users, links between these blogs, and each of the blog is manually assigned one or more labels from a set of 39 category labels. BlogCat2 is a similar dataset with 88,784 blogs and 60 category labels, and additionally each blog may be associated with a subset of 5,413 tags. For our experiments we will consider links between blogs and associated tags as input, and use the manually assigned category labels as gold-standard for evaluation.

Instead of evaluating a clustering method indirectly using a supervised learning task as done previously on these blog datasets [94, 104], we want to directly compare output clusters directly with human assigned categories. However, on a large, noisy dataset with many possible multi-category assignments, it may not be fruitful to compare all clustering and category assignments at once—many of the possibilities can be considered correct and the manually assigned categories may be deficient. Here we will tease out some of these kinds of noise by evaluating a clustering method on one *pair* of categories at a time, instead of the entire dataset.

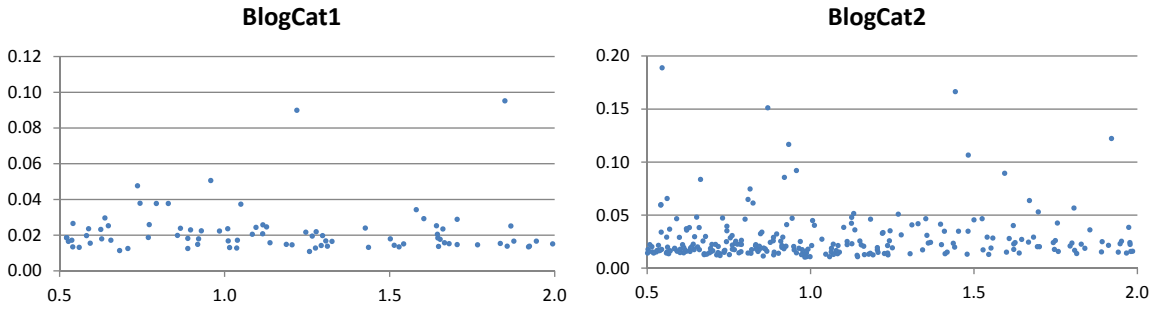
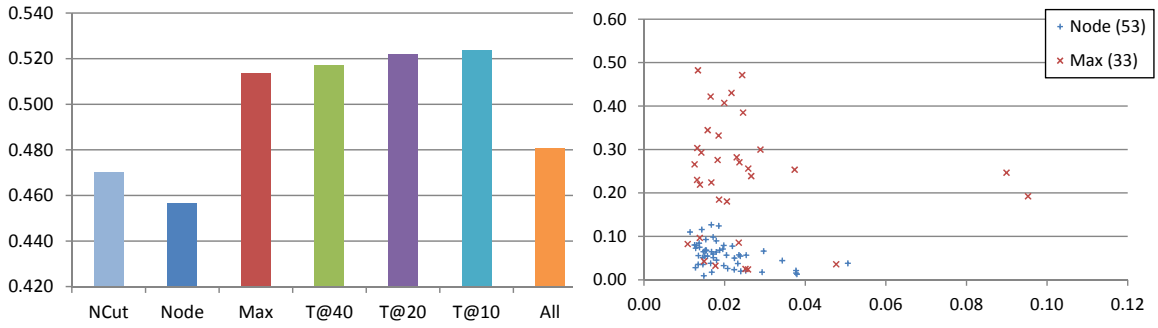



Figure 7.3: Summary statistics for the BlogCatalog datasets. Each dot on the chart is a category-pair dataset. The x-axes correspond to the size ratio between the two categories, and the y-axes correspond to the ratio of instances that belong to both categories.

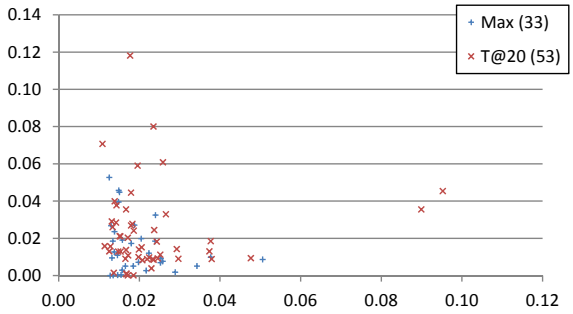
We want to focus on cases where there are actual mixed membership instances, so we select category pairs where when instances belonging to them are pooled together, 2% to 70% of them are mixed membership (belong to both categories). The 70% cap is so there is enough signal there from either category to "guide" the clustering method in separating the data according to the selected categories. We also filter category pairs based on the size ratio between the two categories so that the number of instances in the larger of the pair is at most twice that of the smaller one. We end up with 86 category-pair datasets for BlogCat1 and 158 for BlogCat2. Figure 7.3 plots the pair size ratio against the ratio of mixed membership instances for these datasets. Note that most category pairs have only a small percentage of mixed membership instances. For each



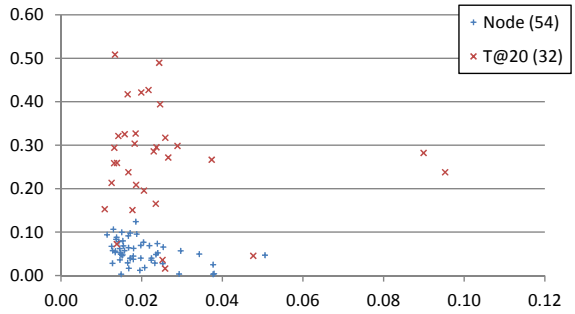

(a) BlogCat1 overall F1




(b) BlogCat1 Node vs. Max F1




(c) BlogCat1 Max vs. T@20 F1




(d) BlogCat1 Node vs. T@20 F1

Figure 7.4: BlogCat1 dataset results. (a) is the overall F1 score averaged over all category pairs, and (b)–(d) are detailed per-category pair charts for detailed comparison of two specific methods.

category-pair dataset, the same method is run 10 times (since all methods involve some random initialization) and the average F1 score of runs are reported.

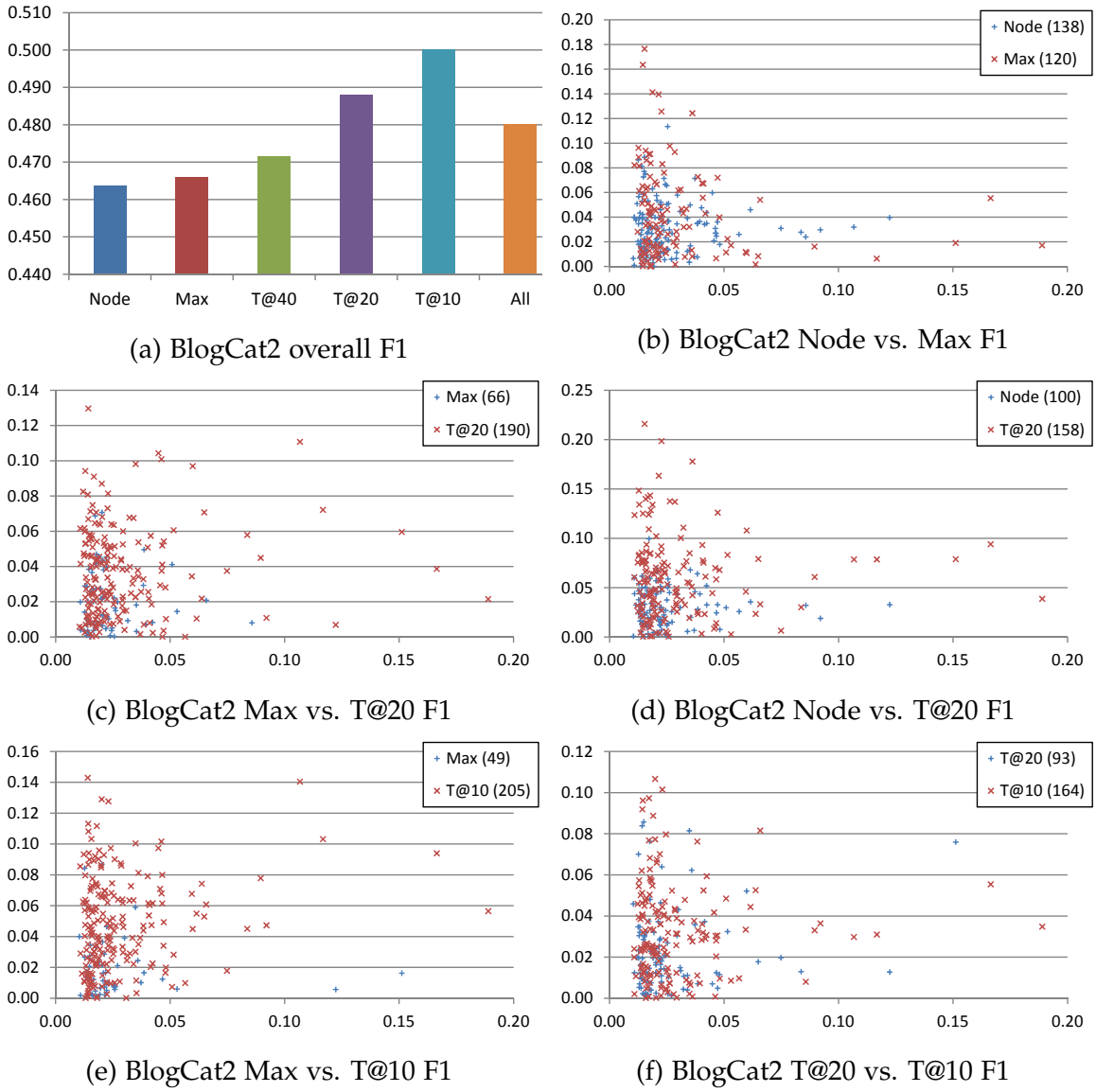
For BlogCat1, we simply use the blog network as input. The overall F1 scores averaged over 86 datasets are shown in Figure 7.4a. An interesting observation here is that not only does edge clustering in general do better than node clustering, even Max is able to outperform both NCut and Node, which suggests that edge clustering may be better than node clustering even on single membership clustering tasks.

Figures 7.4b–7.4d are detailed comparison between two specified methods that require some explanation. Each marker on the plot correspond to one category-pair dataset, and the color and shape of the marker shows which method outperforms the other method, according to the legend in the upper right corner. The legend additionally shows the number of times the method outperforms the other in parentheses. The

x-axes correspond to the ratio of instances in the category pair that have membership to both categories, and the y-axes correspond to the margin by which the winning method outperforms the other on the category pair. For example, in Figure 7.4b, we can see that although Node outperforms Max on more category pairs (53), when Max outperforms Node it is often by a large margin, versus where the winning margins of Node are quite low, indicating that there is very little utility in choosing Node over Max. A general observation of Figures 7.4b–7.4d is that $\text{PIC}_\text{E}$ with an appropriate labeling threshold is almost always better than single membership methods not just in terms of overall performance, but even on a per-dataset scale, and especially for datasets with a higher mixed membership ratio.

For BlogCat2, we want to take advantage of the additional tag information. To test $\text{PIC}_\text{E}$'s performance on general features (not just networks) and even a mixture of different features, the BlogCat2 dataset input contains both links between blogs and tags associated with each blog. Each instance is a feature vector of both tags and links for each blog, which can be interpreted as a bipartite graph and transformed into a BFG as described in Section 7.2. The results for BlogCat2 are shown in Figure 7.5, using the same types of plots as BlogCat1. Note that there are no NCut results for BlogCat2 since NCut does not take general feature vectors as input.

Unlike BlogCat1, where the overall edge clustering methods in Figure 7.4a are rather "flat" with respect to the label assignment threshold parameter p (except for All), the overall result in Figure 7.5a shows a trend for a specific node label assignment parameter p. This trend can be further examined in the detailed comparisons in Figures 7.5b–7.5f, showing that for most category pairs (a) edge clustering method outperform node clustering methods, (b) the methods with high "win" margins are edge clustering methods, and (c) edge clustering methods do better on category pairs with higher mixed membership ratios. In addition, BlogCat2 shows that for certain datasets tuning p is important




(a) BlogCat2 overall F1

(b) BlogCat2 Node vs. Max F1

(c) BlogCat2 Max vs. T@20 F1

(d) BlogCat2 Node vs. T@20 F1

(e) BlogCat2 Max vs. T@10 F1

(f) BlogCat2 T@20 vs. T@10 F1

Figure 7.5: BlogCat2 dataset results. (a) is the overall F1 score averaged over all category pairs, and (b)–(f) are detailed per-category pair charts for detailed comparison of two specific methods.

for an edge clustering approach to output cluster labels that match the ground-truth categories.

## 7.5 Related Work

Palla et al. [83] emphasized the importance of recognizing *overlapping communities* in naturally occurring network data instead of just disjoint communities, and proposed a community discovery method that uses cliques in the graph as the basic structure for inferring communities. The time complexity of the method is exponential to the number of edges in the graph; therefore, even with a small exponent, it does not scale to large datasets.

Mixed membership stochastic blockmodels [6] are a probabilistic method that models both the pairwise presence of links between objects in a network and a global "block" structure that indicates the interaction between clusters. This method provides a probabilistic framework and has the ability to learn parameters (e.g., the interaction between two clusters) and generate random networks based on these parameters. However, since each of the $n^2$ possible edges are a random variable in these models, they are in general not scalable to larger datasets.

EdgeCluster [94] is method for finding the *social dimensions* of a social network graph. Similar to our proposed method, it first construct an edge-centric graph from the social network, where edges become nodes and nodes become edges. Then a modified version of k-means, which is efficient for sparse graphs, is used to produce clusters that corresponds to social dimensions. The nodes of the original graph then are assigned weights along these social dimensions based on its incident edges. Then analysis and predictions can be done based on these social dimensions. The goal of our work is different in that we want to produce clusters that represent real communities, rather than a transformed feature space (though we can also use it for representing such social dimensions).

Correlational learning [104] aims to discover overlapping social groups in social networks supplemented with *tags* (or generally, labels specifying user interest) by first performing a singular value decomposition on the user-tag matrix, and the left and right singular vectors associated with the largest singular values (except the principle singu-

lar vector) are used as features in the latent space for users and tags, respectively. Then the similarity between two user-tag edges are defined as a linear combination of the similarity between the two users and the two tags. Finally, the EdgeCluster $k$-means algorithm [94] is used to discover the social groups given the edge similarities. A drawback of this approach is that SVD computation is in general $O(mn^2 + n^3)$ where $m$ and $n$ are the number the users and tags, whereas BFG integrated with PIC is $O(|E|)$ where $|E|$ is the number of edges in the input graph. In addition, the proposed approach is able to cluster network data and data with arbitrary feature vectors.

Here edge clustering is used as an intermediate step to provide mixed membership clustering. However, if the data is in the form of a graph where nodes are entities and edges are relationships between entities (e.g., social networks, family trees, relational databases, etc.), then edge clustering can be viewed as the clustering of relationships, which is itself an interesting and useful task. While not much work has been done on clustering relationships in social networks (such a task may be even more difficult to evaluate then clustering of entities in social networks), unsupervised methods such as LDA has been applied to the task of discovering *selectional preferences*, the clustering of entity-relationship-entity triples on relational data [87].

# Chapter 8

# Gaussian Kernel Random Walks

## 8.1 Introduction

Random walk methods like PIC (Chapter 2) and MRW (Chapter 4) have runtime and storage requirements linear to the number of edges; therefore, while they are efficient when the graph, or the manifold, is sparse, they do not scale if the graph is dense; i.e., $|E| \approx |V|^2$, where $|E|$ and $|V|$ are number of edges and nodes, respectively.

One approach to the dense manifold problem is to preprocess the graph so it is no longer dense [22, 22, 31, 38, 68, 103, 107, 110], e.g., by constructing a k-nearest neighbors (kNN) graph from the original. Another (arguably more elegant) approach is using an *implicit manifold* (IM), introduced in Chapter 6.

The Gaussian kernel (GK) is a widely-used similarity function and data manifold [77, 91, 114]. It is useful in tasks where a notion of a "neighborhood" on a continuous space is desirable, and the "size" of the neighborhood can be controlled by tuning the Gaussian bandwidth parameter $\sigma$. A notable advantage of having such neighborhoods is that it allows classification and clustering of non-linearly separable data. However, GK manifolds are dense, and currently there is no known sparse decomposition that reconstructs a GK manifold exactly. As a step towards the goal of opening up GK to

many efficient and effective random walk-based learning methods, and more generally to other random walk-based tasks such as ranking [53,82] and retrieval [59], in this paper we present two new *approximate* GK implicit manifolds for continuous data, and analyze their effectiveness on several synthetic and real datasets.

## 8.2 Gaussian Kernel Hilbert Space

In the context of machine learning, kernel-based methods extend linear algorithms to work with non-linear problems by transforming the data to a high- or infinite-dimensional space where only inner product computation is required. Here we are interested in working with the Gaussian kernel (GK), defined as:

$$K(\mathbf{x}, \mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}} \tag{8.1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are vectors (data points in vector space) and $\sigma$ is the "neighborhood width" of the kernel [77, 91, 114]. The $\sigma$ parameter controls the "size" of a "neighborhood" in the manifold space and the kernel is shift- and rotation-invariant (and thus well-suited for data with continuous features like physical space, time, and colors). Because a GK similarity manifold is usually dense and we want to use it on a number of random walk-based learning methods via IM, we seek an sparse, explicit representation of the data instances in the Gaussian kernel Hilbert space (GKHS); i.e., we seek $F$ so that the $i$-th row of $F$ is the representation of example $i$ in the RKHS. Then $FF^T$ can be used in the IM as in Equation 6.6. Currently no such sparse finite decomposition of a GKHS similarity matrix is known. In the following sections we present two approximate GKHS implicit manifolds.

## 8.3 Random Fourier Basis (RF)

One way to approximate an GKHS is by using random Fourier bases in the form $\cos(\omega^T \mathbf{x} + b)$ where $\omega \in \Re^d$ and $b \in \Re$ are random variables, drawn from the Fourier transform $p(\omega)$ of the Gaussian kernel $K$ and the interval $[0, 2\pi]$, respectively. The inner product of two points $\mathbf{x}$ and $\mathbf{y}$ transformed this way is an unbiased estimate of $K(\mathbf{x}, \mathbf{y})$. This transformation has been proposed to provide random features for large-scale kernel machines [86].

This random Fourier (RF) transformation can be interpreted as first projecting a point onto a random direction in $\Re^d$ (making RF rotation-invariant) and passing it through a sinusoidal function with a random bandwidth drawn according to $\sigma$, and then "sliding" the sinusoidal function by a random amount (making it shift-invariant). We can construct a new feature vector $\mathbf{x}'$ for each $\mathbf{x}$ where each element in $\mathbf{x}'$ is the result of an i.i.d. RF transformation. As the dimension of $\mathbf{x}'$ increases, the error $p(|K(\mathbf{x}, \mathbf{y}) - \alpha \mathbf{x}' \cdot \mathbf{y}'|)$ goes to zero, given the appropriate normalizing constant $\alpha$ [86]. This means that we can use the product of a matrix of RF features with its transpose to approximate the GK similarity matrix $A$, and as long as the matrix of RF features is sparse, it is IM-compatible.

We can explicitly construct a Gaussian RKHS approximated with random Fourier features as follows. Given a matrix $X$ with $n$ rows corresponding to data points in features vector space and $m$ columns corresponding to features, we define a matrix $R$ with $n$ rows and $l$ columns. Let $R_{i,j} = \hat{K}(X_i, X_j)$, then $S = RR^T$ yields a similarity matrix $S$ where $S_{i,j}$ is the approximated Gaussian kernel similarity between data points $i$ and $j$. $R$ is constructed as follows [86]:

1. Draw $l$ i.i.d. samples $\omega_1, ..., \omega_l$ from $p$ ($\omega \sim \frac{1}{\sigma}\mathcal{N}(0, 1)$).
2. Draw $l$ i.i.d. samples $b_1, ..., b_l$ from uniform distribution on $[0, 2\pi]$.
3. Let $R_{ij} = \sqrt{2/l}[\cos(\omega_j^T X_i + b)]$.

The space and time complexities of constructing a RF IM are both simply $l$, the number of random Fourier features drawn.

## 8.4 Taylor Features (TF)

Another way to approximate GKHS is by a truncated Taylor expansion of the exponential function. Since GK is in the form of an exponential function, it can be expression as an infinite sum of polynomial functions via Taylor expansion. Explicit construction of an approximate GKHS using truncated Taylor expansion has been proposed for efficient Gaussian kernel large-margin learning [106] [25]; here we will re-derive the approximation and adapt it to the IM framework. We can rewrite the Gaussian kernel from Equation 8.1 as:

$$K(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}} = e^{-\frac{\|x\|^2}{2\sigma^2}} \cdot e^{-\frac{\|y\|^2}{2\sigma^2}} \cdot e^{\frac{\langle x,y\rangle}{\sigma^2}} \tag{8.2}$$

The construction for the first and second terms are straight forward, so we will focus on the third term. Using Taylor expansion of the exponential function about $0$ we have:

$$e^{\frac{\langle x,y\rangle}{\sigma^2}} = \sum_{k=0}^{\infty} \frac{1}{k!} \left( \frac{\langle x, y\rangle}{\sigma^2} \right)^k \tag{8.3}$$

and expanding $\langle x, y\rangle^k$ we get:

$$\langle x, y\rangle^k = \left( \sum_{i=1}^{m} x_i y_i \right)^k = \sum_{j \in [m]^k} \left( \prod_{i=1}^{k} x_{j_i} \right) \left( \prod_{i=1}^{k} y_{i_j} \right)$$

where $j$ enumerates all possible selections of $k$ elements of $\mathbf{x}$ or $\mathbf{y}$ as per the definition of raising a sum of $m$ terms to the $k$th power. Plugging this back into Equation 8.3 we get:

$$\sum_{k=0}^{\infty} \frac{1}{k!} \left( \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\sigma^2} \right)^k = \sum_{k=0}^{\infty} \frac{1}{\sigma^{2k} k!} \sum_{j \in [m]^k} \left( \prod_{i=1}^{k} \mathbf{x}_{j_i} \right) \left( \prod_{i=1}^{k} \mathbf{y}_{i_j} \right)$$

and plugging this back into Equation 8.2 we get:

$$\begin{aligned} e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}} &= e^{-\frac{\|\mathbf{x}\|^2}{2\sigma^2}} \cdot e^{-\frac{\|\mathbf{y}\|^2}{2\sigma^2}} \cdot \sum_{k=0}^{\infty} \frac{1}{\sigma^{2k} k!} \sum_{j \in [m]^k} \left( \prod_{i=1}^{k} \mathbf{x}_{j_i} \right) \left( \prod_{i=1}^{k} \mathbf{y}_{i_j} \right) \\ &= \sum_{k=0}^{\infty} \sum_{j \in [m]^k} e^{-\frac{\|\mathbf{x}\|^2}{2\sigma^2}} \frac{1}{\sigma^k \sqrt{k!}} \left( \prod_{i=1}^{k} \mathbf{x}_{j_i} \right) e^{-\frac{\|\mathbf{y}\|^2}{2\sigma^2}} \frac{1}{\sigma^k \sqrt{k!}} \left( \prod_{i=1}^{k} \mathbf{y}_{i_j} \right) \end{aligned}$$

Therefore an explicit feature representation $\phi$ for each $k$ and $j$ is:

$$\phi_{k,j}(\mathbf{x}) = e^{-\frac{\|\mathbf{x}\|^2}{2\sigma^2}} \frac{1}{\sigma^k \sqrt{k!}} \prod_{i=1}^{k} \mathbf{x}_{j_i} \tag{8.4}$$

where $K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = \sum_{k=0}^{\infty} \sum_{j \in [m]^k} \phi_{k,j}(\mathbf{x}) \phi_{k,j}(\mathbf{y})$, and $[m]^k$ denotes the set of permutations of size $k$ on $m$ objects. An obvious problem with putting this formulation into practice is the summing up of infinite terms indexed by $k$; however we can approximate the sum by restricting to $k \leq d$, thus the Gaussian kernel is approximated by polynomials of degree $d$:

$$\hat{K}(\mathbf{x}, \mathbf{y}) = \sum_{k=0}^{d} \sum_{j \in [m]^k} \phi_{k,j}(\mathbf{x}) \phi_{k,j}(\mathbf{y})$$

Following [25], we will call each $\phi_{k,j}(\mathbf{x})$ in Equation 8.4 a *Taylor feature* of $\mathbf{x}$ and $d$ controls the quality of the approximation. Results on an error bound of this approximation can be found in [25].

**Taylor Feature IM Construction** Next we will see how we can represent this explicit construction as a series of sparse matrix multiplications for incorporation into the implicit manifold framework in Chapter 6.

Given a matrix $X$ with $n$ rows corresponding to data points in features vector space and $m$ columns corresponding to features, we define a matrix $P$ with $n$ rows and $\sum_{k=0}^{d} m^k$ columns. Let $P_{i,j} = \hat{K}(X_i, X_j)$. Then $S = PP^T$ yields a similarity matrix $S$ where $S_{i,j}$ is the approximate Gaussian kernel similarity between data points $i$ and $j$. We can simply plug in $PP^T$ into the methods described in Section 6.5; alternatively, we can decompose the matrix of Taylor features $P$ as:

$$P = AFB$$

Let $X$ be the original feature matrix, $n$ be the number of data points, and $l$ be the number of Taylor features. Then $A$, $F$, and $B$ can be constructed as follows:

- $A$ is an $n \times n$ sparse diagonal matrix and $A_{i,i} = e^{-\frac{\|X_i\|^2}{2\sigma^2}}$.
- $F$ is an $n \times l$ matrix and $F_{i,\xi(k,j)} = \prod_{i=1}^{k} \prod_{j \in [m]^k} x_{j_i}$ where $\xi$ is an indexing operator that "flattens" the indexing over each polynomial degree $k$ and its associated multinomial combinations $j$.
- $B$ is an $l \times l$ sparse diagonal matrix and $B_{\xi(k,:),\xi(k,:)} = \frac{1}{\sigma^k \sqrt{k!}}$.

We make the following notes on the decomposition of $P$. (1) $F$ is the "base" Taylor feature matrix and contains the values of different combinations of multinomials for each polynomial degree $k$. (2) $A$ performs a scalar modification on each data point and can be thought of as a weighting on data points. (3) $B$ performs a scalar modification on each Taylor feature and can be thought of as a weighting on features. (4) Once $F$ and $B$ are constructed, when running learning algorithms, we only need to use the part of the matrices that correspond to the desired degree of approximation $d$; i.e., we never need to reconstruct these matrices if the degree of approximation we want is the same or lower

than what we already have. (5) The Gaussian kernel bandwidth parameter $\sigma$ affects only the sparse diagonal matrices $A$ and $B$. This means we can adjust $\sigma$ efficiently without having to reconstruct all components of $P$.

The space complexity of a TF IM is determined by the number of non-zero entries in $X$; here we want to know, for each data point $\mathbf{x}$, how many non-zero entires are in $F$ given an approximation parameter $d$. The number of non-zero entries in $F$ of a data point $\mathbf{x}$ is related to the product of the permutations (with replacement) of its feature values (see Equation 8.4). For a data point with $c$ non-zero feature values for a particular polynomial degree $k$ in the approximation, the number of non-zero entries $nz(\mathbf{x})$ can be bounded by $\frac{(c+d)!}{d!c!}$. Note that the number of non-zeros $nz(\mathbf{x})$ depends on $c$ and $d$ but *not* on the size of the dataset $n$ or the size of the feature space $m$. This means that with a fixed $d$ and a fixed max non-zero feature value count $\hat{c}$, the density of $F$ is *linear* in the size to $X$, hence TF IM is practical for datasets with a small $\hat{c}$.

The time complexity of the constructing a TF IM is same as its space complexity, because each of the Taylor features can be calculated in constant time based on already computed values (also noted in [25]). Specifically:

$$\phi_{k,j}(\mathbf{x}) = e^{-\frac{\|\mathbf{x}\|^2}{2\sigma^2}} \frac{1}{\sigma^{k-1}\sigma\sqrt{(k-1)!k}} co(k,j)\mathbf{x}_{j_k} \prod_{i=1}^{k-1} \mathbf{x}_{j_i}$$

Note that, unlike Equation 8.4, we allow $j$ to enumerate over all multinomials (combinations of $k$ coordinates) and use a multinomial coefficient function $co(k,j)$.

## 8.5 Experiments

To test the feasibility of this approach, we compare the classification accuracy of two representative random walk-based SSL methods on the two proposed approximate GKHS IMs. In addition, we report the accuracy of a linear support vector machine (SVM) clas-

sifier on the same dataset as a reference, as well as a (non-approximate) GK (which has $n^2$ costs).

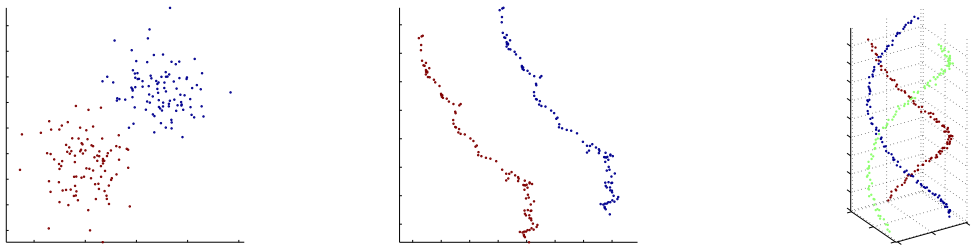
The first SSL methods we use is MultiRankWalk (MRW), a simple and effective random walk classifier that is representative of a number of SSL methods that can be roughly described as label propagation methods via random walk with restart (RWR) [42, 65, 111]. See Algorithm 4 for the implementation we use. For a baseline we will use the cosine similarity IM and we will also explicitly construct the full (size $|V|^2$) GKHS manifold as a performance upper bound. The cosine IM, random Fourier IM, Taylor feature IM, and full GKHS versions of MRW are called $\text{MRW}_{\text{COS}}$, $\text{MRW}_{\text{RF}}$, $\text{MRW}_{\text{TF}}$, $\text{MRW}_{\text{GK}}$, respectively.

The second SSL method we use is the harmonic functions method (HF) [114], another simple and popular classifier that is representative of a number of SSL methods that can be roughly described as label propagation methods via reverse random walk with "sink nodes" [9, 13, 71]. The particular implementation we use is also called a weighted-voted relational network classifier [71]. *Class mass normalization* [114] is an optional heuristic that often improves the output of HF; in our experiments we run HF both with and without it and report only the best of these two. The four IM versions of HF are called $\text{HF}_{\text{COS}}$, $\text{HF}_{\text{RF}}$, $\text{HF}_{\text{TF}}$, $\text{MRW}_{\text{GK}}$.

For SVM we use LIBLINEAR [34] using the L2-regularized L1-loss support vector classification setting with a bias of 1. All together we have 9 classifiers for the experiments.

**Synthetic Data** We constructed three types of spatial synthetic datasets, shown in Figure 8.1. 8.1a are points generated from Gaussians of two uniformly random means in $\mathfrak{R}^2$ within a unit interval; 8.1b are two copies of a set of points generated with a Markov random walk in $\mathfrak{R}^2$ biased in a random direction, with one copy "shifted" in a direction orthogonal to the biased direction from the other copy; 8.1c are points that form three

"noisy" spirals in $\Re^3$ that together form a helix-like structure—each point deviates independently from the perfect spiral by a small random amount. The colors indicate class labels.



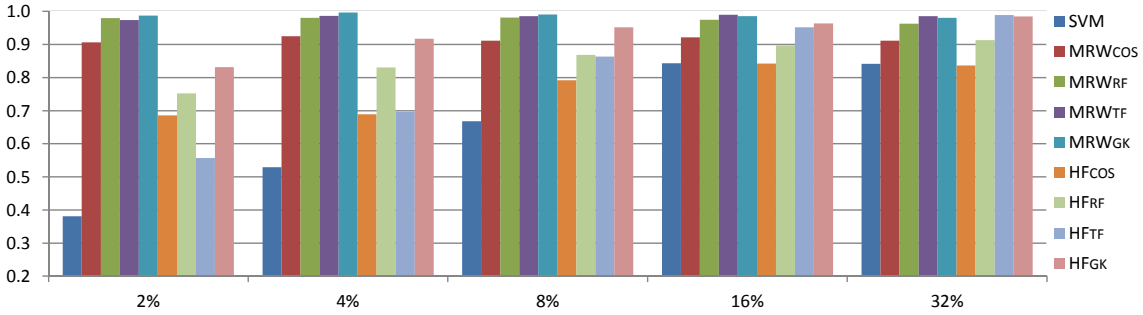
(a) 2-Gaussians example (b) 2-random walks example (c) 3-spirals example

Figure 8.1: Examples of randomly generated synthetic datasets. Colors indicate class labels.

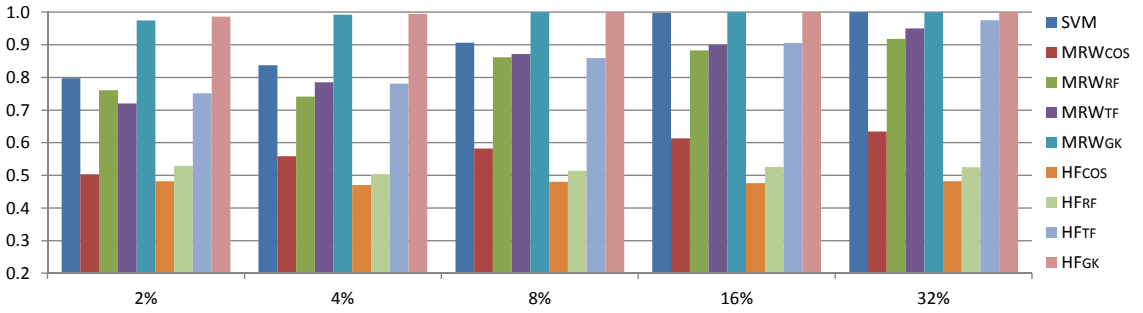
Each type of synthetic dataset is designed to test different problem conditions. In 8.1a, the two classes can potentially be very close and even overlap each other; to accurately separate the two classes the SSL method needs to make full use of the unlabeled instances. In 8.1b, although the points are linearly separable by a large margin, the manifold must be rotation-invariant because the random walk can be biased in any direction. In 8.1c, the points are not linearly separable.

We generate 100 datasets for each type of synthetic data, and each dataset contains 100 points per class. The restart parameter for all MRW methods is set at $\alpha = 0.05$. For both the random Fourier and Taylor feature IMs we set parameters that control the approximation complexity at 16 features per instance. The classification accuracy result is shown in Figure 8.2, where the best result from a parameter sweep of $\sigma$ is reported. For each dataset a specified percentage of instances are used as the training set and the rest are the test set.

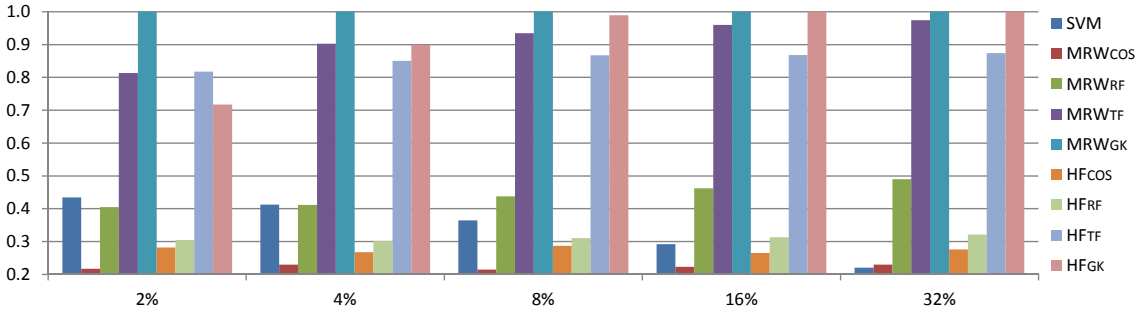
We make a number of observations based on Figure 8.2. (1) As expected, the full GKHS manifold are always as good or better than the approximations. (2) MRW in general outperforms HF when the amount of training data is small as observed previously




(a) 2-Gaussians F1 scores vs. training set size




(b) 2-random walks F1 scores vs. training set size




(c) 3-spirals F1 scores vs. training set size

Figure 8.2: Classification accuracy on three types of synthetic datasets. The vertical axis corresponds to the F1 score and the horizontal to the training set size.

on network datasets [65]. (3) As expected, SVM does poorly on noisy dataset when the training set is small (Figure 8.2a), and on datasets that are strongly non-linear separable (Figure 8.2c). (4) RF IM does not work as consistently as TF IM; RF IM works well on the Gaussian data, fails to work on the spirals data, and works only with MRW on the random walks data. (5) The only method that work consistently across the data types without using the computationally expensive full GKHS manifold is $\text{MRW}_{\text{TF}}$.

**On GeoText Data** We test the proposed methods on a challenging real dataset. The GeoText dataset [32] consists of a week's worth of about 380,000 microblog posts from 9256 users in the United States, with the user's geolocation in latitude and longitude, collected from the microblog website Twitter. Given a training set of user messages and locations, the task is to predict an unseen user's location given his or her message text. To do well on this task, a method must leverage the textual similarity between user's posts and the geolocation proximity between users. Eisenstein et al. [32] trained a latent variable geographic topic model that outperformed several strong baseline methods on both a geolocation prediction task and a 4-label region classification task.

Since there is no explicit modeling, or even representation, of continuous manifolds such as geolocations using IMs, there is not a straightforward way to do geolocation prediction using GK IMs. However, we can do the region classification task by propagating region labels between users that are nearby each other and users with similar posts. To do this, we want to construct an IM that uses both the text data and the geographic data. Let $W$ be a square word-similarity matrix between users and G be a square geo-similarity between users, then $D_W^{-1} W D_G^{-1} G$, where $D_W(i,i) = \sum_j W(i,j)$ and $D_G(i,i) = \sum_j G(i,j)$, specifies a two-step transition matrix where the first step the transition probability is according to the word-similarity, and the second step according to the geo-similarity. This is a simple way to compose two types of similarity features in IMs. By replacing $W$ and G with the appropriate IMs (e.g., cosine IM and TF IM) and substituting the entire composition for $A$ in the desired SSL method (e.g., Step 6 of Algorithm 4), we end up with a method that efficiently finds similar groups of users via geolocation and textual similarities. Prior work on this dataset [32] used a fixed training (60%), develop (20%), and test (20%) split; here we use a random 60% training and 40% test split for each of 20 trials and report the average classification accuracy in Table 8.1.

We see that on this task $\text{MRW}_{\text{TF+}}$ is competitive with the much more complex state-of-the-art geographic topic model, and beats all other baselines. We also see that $\text{MRW}_{\text{TF+}}$

| Method | Accuracy |
|---|---|
| Geographic topic model | **0.58** |
| Mixture of unigrams | 0.53 |
| Text regression | 0.41 |
| Supervised LDA | 0.39 |
| K-nearest neighbors | 0.37 |
| Most common class | 0.37 |
| MRW (text) | 0.52 +- 0.0091 |
| $\text{MRW}_{\text{TF+}}$ (geo+text) | **0.57** +- 0.0078 |

Table 8.1: GeoText results from prior work and MRW with composed IMs. Accuracy is the micro-averaged classification accuracy, and Time is the wall-clock time in seconds. Results above the double line are reproduced from prior work [32] for comparison. MRW uses only the word-similarity and $\text{MRW}_{\text{TF+}}$ uses both word- and geo-similarity.

is able to take advantage of the geolocation feature to outperform a mixture of unigrams, text regression, and MRW (text only). The wall-clock running time is 0.12 seconds on average.

Though our experiments show TF to outperform RF with these SSL methods, an important thing to note is that unlike RF, the approximation error made by TF is *dependent* of where the instance lie in the original space. Because a Taylor expansion is *about* a particular point (the origin in this case), the error increases as we get further away from the point of expansion. The following bound from Cotter et al. [25] bounds the error in terms of feature vector lengths:

$$|K(\mathbf{x}, \mathbf{y}) - \hat{K}(\mathbf{x}, \mathbf{y})| \leq \frac{1}{(d+1)!} \left( \frac{\|\mathbf{x}\| \, \|\mathbf{y}\|}{\sigma^2} \right)^{d+1}$$

Note the inverse correlation between the vector lengths and $\sigma^2$. In addition to re-centering the data around the origin, we may need to adjust $\sigma$ as to balance between the right bandwidth parameter and an adequately large "effective radius." This also implies that a $\sigma$ that works well with a full GK manifold may need to be re-tuned for TF IM.

# Chapter 9

# Conclusion and Future Work

## 9.1 Conclusion

We showed that *power iteration clustering* (PIC) is a general and scalable alternative to graph-based clustering methods such as spectral clustering, and that *MultiRankWalk* (MRW) is a scalable and effective graph-based semi-supervised classification method when given only a small amount of training instances—and where those training instances are located in the graph structure may greatly affect the classification performance.

We presented *implicit manifolds* (IM), a technique and a framework for extending iterative graph-based learning methods such as PIC and MRW to general data in vector space for a class of similarity functions; in particular, we have shown both PIC and MRW to work effective with text data via IM on applications such as document clustering and noun phrase categorization.

We then further extended PIC to do mixed membership clustering via edge clustering, and we also showed that where an exact IM is not feasible, as is the case with Gaussian kernels, we can use an approximate IM for random walk learning tasks.

The ideas and methods presented in this thesis lay a groundwork for many interesting extensions and applications; in sections to follow we list a few of them.

## 9.2 PIC Extensions

PIC is a basic method that can be extended in many ways. We have described multi-dimensional PIC for dealing with a large k (Section 2.7), PIC for general feature vector data via implicit manifolds (Section 6.4), and PIC for mixed-membership clustering (Chapter 7). Here we will describe additional extension ideas that are also compatible with all the existing extensions.

### 9.2.1 Initialization

Similar to other learning algorithms that involve random initialization (e.g., k-means, Gibbs sampling, expectation-maximization, etc.), the initial vector $\boldsymbol{v}^0$ of PIC maybe have a significant effect the clustering result. There are two items to considering when initializing PIC: (a) instances in different clusters should be as far away from each other as possible, and (b) for multi-dimensional PIC, the initial vectors $\boldsymbol{v}_i^0$ should be such that the final vectors $\boldsymbol{v}_i^t$ are linearly independent and as close to orthogonal as possible. These considerations pose a chicken-and-egg problem: if we know the optimal initialization then we have solve the problem already. However, if we make certain assumptions about the data based on prior knowledge, we can tweak the otherwise uniformly random initial vectors and achieve better clustering results.

**Normal Initialization** Mentioned in connection to *random projection* in Section 3.6, initializing PIC according to a Gaussian distribution may help to preserve the relative pair-wise distances between points in the PIC embedding, according to prior results on random projection [16, 26, 99].

**Degree Initialization** Initializing PIC such that each element of the initial vector is set to the degree of its corresponding node (i.e., $v^0(i) = \sum_j A(i,j)$) can lead to better-separated clusters if the average degree of each cluster is different. This average degree heterogeneity assumption can be inferred from other assumptions; for example, if can assume the average degree is correlated to cluster size and that the clusters are of different sizes, than the average degree will also differ between clusters. An additional benefit of degree initialization is that it leads to faster convergence. This is because at convergence elements of the PIC vectors corresponding to the same cluster will have the same value, and concentrating the high values in nodes with higher degree will allow them to more quickly "propagate" the values to nodes with lower degrees.

**Skewed Initialization** To better separate a larger number of clusters with one or few PIC vectors, we want to further "skew" the distribution of the initial values so that the average degree of a cluster is not likely to be the same. An example of a skewed initialization procedure would be: (1) run PageRank on $A$ and create a rank vector $\mathbf{r}$ such that $\mathbf{r}(i) = 1$ if $i$ has the highest PageRank value, and $\mathbf{r}(j) = 2$ if $j$ has the second highest PageRank value, and so on, (2) pick $h$ highest ranked nodes (e.g., $h = 32$), (3) set $v^0(i) = 2^{h-i}$ if $\mathbf{r}(i) < h$, and (4) set the rest of $v^0$ to zero.

### 9.2.2 Stopping

In Section 2.5 we introduced a simple and intuitive criterion for stopping PIC iterations—"acceleration", based on the assumption that there is a noticeable *eigengap* between the eigenvalues of the normalized affinity matrix $W$. PIC can be used with other stopping criteria based on the desired clustering result or other prior assumptions or prior knowledge about the data.

**The Normalized Cut Objective** Since $W$ is directly related to the normalized Laplacian used for Normalized Cut (NCut) [91], it makes sense to use its objective function as a criteria for PIC. At every PIC iteration we can run k-means and calculate the NCut objective; the loop continues if the objective is better than the previous iteration and it terminates when the objective is the same or worse than the previous iteration. Note that while running k-means and calculate the NCut objective at every iteration can be rather expensive for large datasets, this can be made more efficient by initializing k-means using the final centers from the previous iteration and doing so only once every few steps instead of every step.

**Modularity** Given a graph and a number of clusters, *modularity* is the fraction of the edges that fall within the given clusters minus the expected such fraction if the edges were randomly distributed [79]. Like NCut, modularity can be optimized directly using matrix decomposition methods for finding communities in a network [79] and we can also use it as the stopping criterion in the same way we use the NCut objective.

**The Earth Mover's Distance** As noted in Chapter 2 and 3, with proper normalization, the PIC vector $\boldsymbol{v}^t$ can be viewed as the random walk probability distribution t steps *before* $\boldsymbol{v}^0$. We can then calculate the change in this probability distribution between iterations and stop when the change in this probability is low. A well-known method for calculating the distance between two distributions is the *earth mover's distance* (EMD) [89], used in computer vision research for computing distances between grayscale images and color histograms. The idea is that a categorical distribution p has certain amount of "earth" which is to be moved into "holes" with capacity according to a categorical distribution q, over a certain amount of "ground distance". A unit of work is done when transport a unit of earth by a unit of ground distance, and the EMD between two distribution is the least possible amount of work done moving all of the earth from p to q.

There are two problems applying EMD to PIC. First, it is not clear how to assign ground distances between different categories as in image applications (e.g., the ground distance between grayscale pixels can simply be the euclidean or Manhattan distance between their locations in the image). Second, the general computation method for EMD is the Hungarian algorithm, which is $O(n^3)$ and very expensive for larger datasets. A simple solution to these problems is setting the ground distance to 0 between corresponding categories and 1 for everything else.

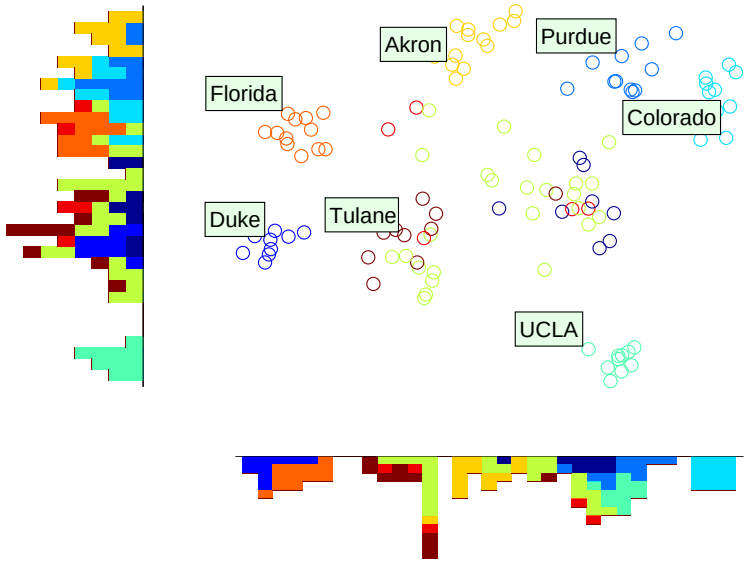
**Variable Stopping Points** In Section 2.5 we show that as the number of iteration increases, the signals from eigenvector $\boldsymbol{e}_i$ disappears from the embedding at a rate of $\left(\frac{\lambda_i}{\lambda_1}\right)^t$; and later in Section 3.6 we show that $t$ is a scaling factor in the diffusion space. These observations imply that, rather than viewing PIC as having a definite stopping point with the "best" embedding, we can view an embedding with a smaller $t$ as fine-grained and one with a larger $t$ as coarse-grained. Since the embedding reveals the structure of the data at different scales, it may be useful to create a multi-dimensional PIC embedding based on different $t$'s, instead of setting a particular stopping point.

## 9.3 Data Visualization

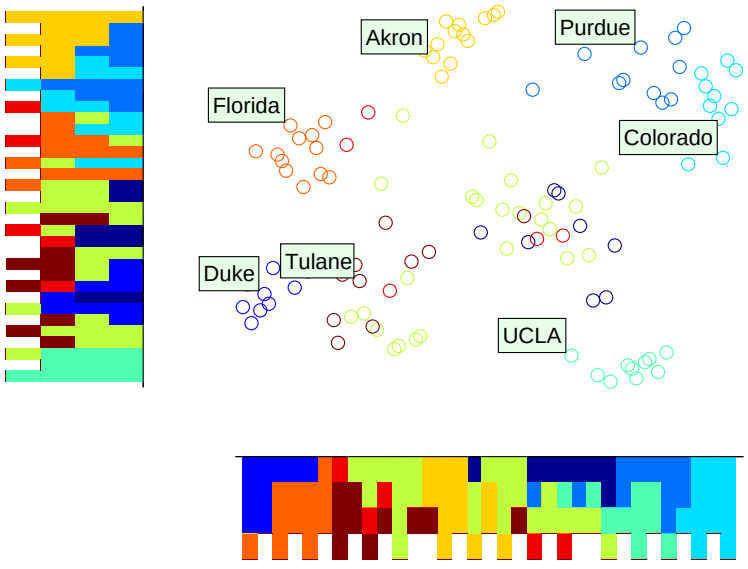
Data visualization is the task of representing data in a high-dimensional space or a complex manifold in a low-dimensional Euclidean space (usually in 2 or 3 dimensions) so that is can be displayed on a computer screen or in print for data exploration, analysis, and representation. An algorithm that projects data onto a low-dimensional space for visualization is typically called a *layout algorithm*, and layout algorithms found in popular graph visualization tools such as JUNG [2] and Gephi [14] are often based on physical simulation of objects. For example, the SpringLayout algorithm in JUNG simulates the input graph as a network of interconnected coil springs, which act as attraction forces between connected nodes. Of course, nodes should also repel one another to avoid form-

ing a single large cluster in the visualization. The simulation of repelling and attracting forces lies at the core of many layout algorithms, and almost all of them are $O(n^2)$ in the number of nodes. Aside from the capability of the graphics rendering hardware, this is the main reason why many of these tools are limited to visualizing a small dataset or a small sample of a large dataset.

A natural application of a low-dimensional PIC embedding is data visualization, and its main advantage over traditional spectral-based dimensionality reduction approach such as PCA (Section 3.4) is again its speed. In addition, with implicit manifolds (Chapter 6) PIC can also be used to visualize other types of data. To avoid data points overlaying each other, we can use simple heuristics to "spread" the data points over available space while preserving the general look of the clusters. Figure 9.1 shows a PIC visualization of the Football dataset with and without "spreading". Note that while Figure 9.1a and 9.1b look very similar and both reveal the structure (football conferences) of the network data, in Figure 9.1b the points are much more evenly spread out, as indicated by the colored histograms, so data points can be accessed individually. For example, the light blue and dark blue points in the top right corner are no longer displayed directly on top of each other.




(a) without "spreading"




(b) with "spreading"

Figure 9.1: A 2D visualization of the Football network data using the PIC embedding. Points are American colleges and colors correspond to football conferences. The x and y axes are overlaid with histograms to show the concentration of points on that dimension. Some colleges are labeled as example instances of the conferences they belong to.

## 9.4 Interactive Clustering

Since a PIC visualization, as described in Section 9.3, is directly derived from the PIC embedding, we can turn this around—the position of objects in a visualization can also be used to "fix" the PIC embedding and thus give better clustering results. Based on this idea, we can imagine an iterative clustering procedure like Algorithm 9.

**Algorithm 9** Interactive PIC

**procedure** $\text{PIC}_{\text{INT}}(A, k)$
  Run PIC($A, k$) and get 2D embedding $E$ and clusters $\mathcal{C}$
  **repeat**
    Display $E$ as points in 2D colored according to $\mathcal{C}$
    Ask user to fix $E$ by dragging points, update colors as needed
    Update $E$ according to dragged points
    Run PIC($A, k$) with $E$ but "lock" the position of the dragged points
  **until** User no longer make changes
  Save the positions of the dragged points for future use
  **return** $\mathcal{C} = \{C_1, C_2, ..., C_k\}$
**end procedure**

This clustering procedure is *interactive* because of iterative refinement of the clusters based on user input. The proposed method is very different from most prior work on interactive clustering, where the user input is primarily choosing cluster features or cluster membership [47]. Here we combine interactive clustering with data visualization, which may prove to enhance the user experience and provide better clustering results per unit of user input.

## 9.5 Learning Edge Weights for PIC

In Chapter 2 PIC is proposed as an unsupervised clustering method. Some clustering methods are supervised. For example, *constrained clustering* is a class of clustering methods that are supervised via a set of constraints such as *must-link* and *cannot-link* constraints. Unlike semi-supervised classification, no class label is given, but instead the

training data specifies which instances should be in the same cluster and which ones should not.

PIC's main task is to provide a low-dimensional metric embedding for data that lie in a pair-wise similarity manifold for a downstream clustering method (such as k-means) to assign the cluster labels, so it makes sense to replace k-means with one of these constrained clustering methods to handle must-links and cannot-links. However, we can also use these types of training data to re-weight the edges of the input graph if the graph contains multiple types of nodes or edges.

For example, in a social network of people, the edges maybe labeled with edge types such as "friends", "spouse", "siblings", etc.; or perhaps the edges are not labeled, but we have nodes labeled with attributes such as sex and age, which we can use to label the edges with types such as "male-female", "same-sex", "old-young", or "same-generation". With multiple edge types it may be helpful to consider different types of edges of different importance; for example, a "close-friend" edge type may indicate a stronger affinity than a "colleague" edge type.

Suppose we are given a set of pair-wise must-link constraints $P = \{(a_1, b_1), (a_2, b_2), ...\}$ and a set of cannot-link constraints $Q = \{(a_1, b_1), (a_2, b_2), ...\}$ on the nodes of affinity matrix $A$, where the set edges $E$ are composed of edges of different types: $E = E_1 \cup E_2 \cup ...$; $\forall_{i,j} E_i \cap E_j = \emptyset$. Let the row-normalized matrix $W$ be parameterized by $\alpha$ according to the edge types, and recall that the PIC embedding vector at time $t$ is $\boldsymbol{v}^t = W^t \boldsymbol{v}^0$ and $W = D^{-1}A$ where $A$ is the affinity matrix and the diagonal matrix $D = \sum_j A(i,j)$. Then

$$W(i,j) = \frac{A(i,j)}{D(i,i)}$$

and

$$A(i,j) = \exp\left(\sum_k \alpha(k)\phi(i,j,k)\right)$$

where $\phi(i,j,k)$ is a function indicating if the edge $(i,j) \in E_k$. We then have the following objective function:

$$\min_{\alpha} \sum_{(a,b)\in P} (\boldsymbol{v}^t(a) - \boldsymbol{v}^t(b))^2 - \sum_{(a,b)\in Q} (\boldsymbol{v}^t(a) - \boldsymbol{v}^t(b))^2 \tag{9.1}$$

In other words, we want the values of the must-link pairs to be "close" to each other in $\boldsymbol{v}^t$ and the values of the cannot-link pairs to be "far away" from each other. We then take partial derivative of Equation 9.1 with respect to $\alpha$:

$$\begin{aligned}
&\frac{\partial}{\partial\alpha}\left[\sum_{(a,b)\in P} (\boldsymbol{v}^t(a) - \boldsymbol{v}^t(b))^2 - \sum_{(a,b)\in Q} (\boldsymbol{v}^t(a) - \boldsymbol{v}^t(b))^2\right] \\
=&\frac{\partial}{\partial\alpha}\left[\sum_{(a,b)\in P} (\boldsymbol{v}^t(a) - \boldsymbol{v}^t(b))^2\right] - \frac{\partial}{\partial\alpha}\left[\sum_{(a,b)\in Q} (\boldsymbol{v}^t(a) - \boldsymbol{v}^t(b))^2\right] \\
=&\sum_{(a,b)\in P} 2\left[\frac{\partial}{\partial\alpha}\boldsymbol{v}^t(a) - \frac{\partial}{\partial\alpha}\boldsymbol{v}^t(b)\right] - \sum_{(a,b)\in Q} 2\left[\frac{\partial}{\partial\alpha}\boldsymbol{v}^t(a) - \frac{\partial}{\partial\alpha}\boldsymbol{v}^t(b)\right]
\end{aligned}$$

For each $\frac{\partial}{\partial\alpha}\boldsymbol{v}^t(i)$ term in the sum:

$$\begin{aligned}
\frac{\partial}{\partial\alpha}\boldsymbol{v}^t(i) =&\frac{\partial}{\partial\alpha} W\boldsymbol{v}^{t-1}(i) \\
=&\frac{\partial}{\partial\alpha}\sum_j W(i,j)\boldsymbol{v}^{t-1}(i) \\
=&\sum_j \frac{\partial}{\partial\alpha}\left[W(i,j)\boldsymbol{v}^{t-1}(i)\right] \\
=&\sum_j \left[\boldsymbol{v}^{t-1}(i)\frac{\partial}{\partial\alpha}W(i,j) + W(i,j)\frac{\partial}{\partial\alpha}\boldsymbol{v}^{t-1}(i)\right]
\end{aligned} \tag{9.2}$$

The above equation is recursive, but we can also apply the chain rule recursively until $\frac{\partial}{\partial\alpha}\boldsymbol{v}^0(i)$ and compute the gradient iteratively. For multi-dimensional PIC we can simply minimize the same equation over multiple $\boldsymbol{v}^0$'s, or we can replace $(\boldsymbol{v}^t(a) - \boldsymbol{v}^t(b))^2$

in Equation 9.1 with $\left\|v_a^t - v_b^t\right\|^2$ where $v_i^t \in \Re^d$ is the vector for instance $i$ in a $d$-dimensional PIC embedding.

The recursive derivation in Equation 9.2 is similar to the derivation of the objective function for learning edge weights in *supervised random walks* (SRW) [10] for the task of predicting and recommending links in a social network. However, the objective function and the propagation matrix used in the SRW work are quite different from ones presented here; in the SRW work the task is to provide rankings instead of clusters, and the propagation is via forward random walks (with restarts) rather then backward random walks (with sink nodes).

## 9.6 Improving Edge Clustering

We proposed transforming a graph into a bipartite feature graph (BFG) as a general approach to apply graph partition methods such as spectral clustering to mixed membership clustering tasks. We show that a well-suited method is power iteration clustering (PIC), and when appropriately combined with BFG, it is able to show substantial improvement over single membership methods on even moderately mixed membership datasets.

An improvement to the proposed approach is learning the label assignment threshold parameter $p$ based on some statistics or prior knowledge of category distribution. We can also extend this approach to apply BFG transformation to hypergraphs by clustering hyperedges. Lastly, we want to verify the generality of mixed membership clustering via edge clustering on a number of other efficient graph partition methods.

## 9.7 Other Methods for Random Walk Learning on Continuous Data

In Chapter 8 we described methods for random walk learning on the Gaussian kernel Hilbert space. GKHS is just one of many kernel functions that can be applied to data in continuous spaces. For example, we can use apply similar approximation methods to other radial basis kernels such as the Laplacian kernel and Cauchy kernel [86]. If we are interested in kernels that depend on the $L_1$ distance between pair-wise points instead of $L_2$, we can also use methods such as randomly "binning" the points with randomly shifted grids at randomly chosen resolutions, which has been shown to work well with SVM on a number of datasets [86].

The Taylor features method and the random Fourier method described in Chapter 8 and many other random approximation methods such as ones described in Rahimi and Recht [86] approximate a continuous space by partition it with functions drawn from distributions independent of the data. We may improve accuracy on a learning task by drawing random functions according to the data distribution; for example, we may want to set a minimum and maximum bandwidth according to prior knowledge about the data when drawing random Fourier features.

If the dimensionality of the space we want to approximate is large, the complexity of the approximation methods mentioned above may become intractable if we want to keep the approximation error below a certain threshold [25]. In such cases we can first reduce the dimensionality of the data using random projection or locality sensitive hash (Section 3.4) before using a kernel approximation method.

## 9.8 k-Walks

An interesting application of the semi-supervised method MRW is to use it for clustering. The basic idea to use standard k-means algorithm (Lloyd's algorithm), but replace the Euclidean distance functions with random walk probabilities. In each iteration the random walk probabilities from k centers to all the instances is computed using MRW (Algorithm 4)—i.e., k random walks with restart, and the new centers are obtained using k PageRank [82] computations; a basic version is shown as Algorithm 10.

**Algorithm 10** The k-walks algorithm

**procedure** KWALKS($A, k, \alpha, \beta$)
  Random pick cluster centers $\mathcal{M}^0 = \{m_1^0, m_2^0, ..., m_k^0\} \in \mathcal{X}$
  $t \leftarrow 0$.
  **repeat**
    Obtain clusters $C_1^t, C_2^t, ..., C_k^t$ via $MRW(A, \mathcal{M}^t, \alpha)$ using $\mathcal{M}^t$ as seed instances
    **for** each $C_i$ **do**
      Form graph $A_i^t$ as a subgraph of $A$ formed by $C_i^t$
      Compute PageRank $\mathbf{r}_i^t$ on $A_i^t$ with teleport probability $\beta$
      Set new center $m_i^t \leftarrow max_j \mathbf{r}_i^t(j)$
    **end for**
    $t \leftarrow t + 1$
  **until** $\mathcal{M}^t = \mathcal{M}^{t-1}$ or a maximum $t$ reached
  **return** $C_1^t, C_2^t, ..., C_k^t$
**end procedure**

Here we make some observations about this method:

- Since computing k PageRank probabilities is the same as running MRW with a uniform restart vector $\mathbf{r}$, all MRW extensions can be applied (e.g., adapting to general feature vector data via implicit manifolds).
- Assuming the number of iterations per MRW is constant [1], it has the same complexity as that of k-mean, which is generally considered very efficient.

[1] It does not depend on the size of the dataset; rather, it depends on the restart probability and the structure of the data—the eigenvalues.

- Like k-means, initial centers $\mathcal{M}$ can be chosen more cleverly; we can chose them according to prior knowledge, choose them according to a desirable statistical property (e.g., k-means++ [8]), or even fix some of the centers if we want to do a mixed SSL and clustering task.
- Instead of choosing a point as the center in Step 9 we can pick a few points or even a distribution over the points as the center. Using a distribution as the center makes k-walks more analogous to k-means (Lloyd's algorithm).
- Similar to PIC and *kernel* k-*means* [30], k-walks can be viewed as k-means on a random walk embedding; however, unlike these, the embedding changes every iteration according to the current centers and clusters.
- It can be shown that, using points as centers, Algorithm 10 may not converge; we have yet to analyzed and see if it would converge if we use distributions as centers.

| | NCut | NJW | $\text{KWalks}_\text{P}$ | $\text{KWalks}_\text{D}$ |
|---|---|---|---|---|
| UMBCBlog | 0.953 | 0.953 | 0.930 | 0.910 |
| AGBlog | 0.342 | 0.342 | 0.940 | 0.944 |
| MSPBlog | 0.389 | 0.389 | 0.750 | 0.744 |

Table 9.1: Macro-averaged F1 scores on 3 political blog datasets. $\text{KWalks}_\text{P}$ and $\text{KWalks}_\text{D}$ are the k-walks algorithm using point centers and distribution centers, respectively.

A preliminary result comparing k-walks to two spectral clustering methods on the political blog datasets described in Section 4.4.1 is shown in Table 9.1. It shows that k-walks is very competitive with spectral clustering methods UMBCBlog and is able to do well on AGBlog and MSPBlog, both of which the spectral clustering methods fail completely.

## 9.9 Uniform Sampling for Random Walk Learning

Methods such as HF and MRW (Section 4) are generally efficient as long as the graph is sparse because the random walk probabilities for each class can be computed in $O(|E|)$ where $|E|$ is the number of edges. However, what if the number of classes scales up with the number of edges or nodes in the dataset? This is not unlikely if, say, the classes are small communities (e.g., families or student organizations) in a social network. In that case HF and MRW will incur a $O(n^2)$ computational cost, where $n$ is the number of nodes, even if the graph is sparse.

An observation that can be made about multiple biased random walks on graphs (e.g., HF and MRW) is that, if we have $k$ random walks representing $k$ classes, the probabilities of most biased random walks will reach a given node is typically very small—only a few of the walks will have enough probability to compete for the node's membership, and the probability of the rest of them are negligible.

One way to ameliorate this problem is to *sample* the random walks instead of calculating them exactly as in Chapter 4. We can sample the random walks, approximate the probabilities with this sample, and then determine class membership based on the approximated probabilities. To sample the random walks, we can label each biased random walker with its class label, and then simulate a Markov random walk with each of the walkers. Each node will act as an "observer"—it will record the number of times it has been visited by random walkers from each of the class label. Since the amount of memory is limited, especially for large datasets, we cannot record every node visit by every random walker—we need to decide when to record, when not to record, and when to erase an old record if the space limit has been reached, until the algorithm converges.

A common solution is to simply record everything until the space limited has been reached, and then only record visits by classes of higher probabilities based on current records and erase records by classes of low probabilities based on current records if needed. This solution is problematic in that it does not produce an *uniform sample* of all

the visits. The sample is biased toward the initial visits, making it sensitive to random paths taken at the beginning of the simulation and biased toward short paths and away from long ones.

**Algorithm 11** A reservoir sampling algorithm

```
1: procedure RS(a, n, m)
2:     for i = 1 → m do
3:         s(i) ← a(i)
4:     end for
5:     for i = m + 1 → n do
6:         j ← rand([1, i])
7:         if j ≤ m then s(j) = a(i)
8:     end for
9:     return s
10: end procedure
```

For getting a random walk sample we propose to use *reservoir sampling* (RS) [102], an elegant and computationally efficient method for obtaining a uniform sample from a stream of objects. Given an array or stream $\mathbf{a}$, a stopping index $n$, and a sample size $m$, RS returns a uniform sample $\mathbf{s}$ of size $m$ such that an element in $\mathbf{a}$ is in $\mathbf{s}$ with probability $\frac{m}{n}$. A basic RS algorithm is shown as Algorithm 11.

Note that the proposed method for scaling up multiple random walks is very different from sampling methods that scales up a single random walk, such as strategies to prune random walk paths below a certain threshold [20]. For a single random walk, where the application is often the retrieval and ranking of top items, we are interested in computing accurate random walk probabilities for a small subset of nodes with a high likelihood of being reached by the walk; therefore we can safely "prune away" nodes or edges that we predict to have a low likelihood of being reached by the walk. However, for multiple random walks where the application is to classify each item, we need to estimate random walk probabilities for each node (no pruning), but for each node we only need to know which one of the random walks visits it the most—an accurate probability is not necessary.

# Bibliography


[1] Amazon Mechanical Turk. www.mturk.com.

[2] Java universal network/graph framework. jung.sourceforge.net.

[3] Nielsen buzzmetrics. www.nielsenbuzzmetrics.com.

[4] World Gazetteer. world-gazetteer.com.

[5] Lada Adamic and Natalie Glance. The political blogosphere and the 2004 u.s. election: Divided they blog. In *The WWW 2005 Workshop on the Weblogging Ecosystem*, 2005.

[6] Edoardo M. Airoldi, David M. Blei, Stephen E. Fienberg, and Eric P. Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9:1981–2014, 2008.

[7] F. Alimoglu and Ethem Alpaydin. Combining multiple representations and classifiers for handwritten digit recognition. In *The International Conference on Document Analysis and Recognition*, 1997.

[8] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *The 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[9] Arik Azran. The rendezvous algorithm: Multiclass semi-supervised learning with markov random walks. In *The 24th International Conference on Machine Learning*, 2007.

[10] Largs Backstrom and Jure Leskovec. Supervised random walks: Predicting and recommending links in social networks. In *The 4th ACM International Conference on Web Search and Data Mining*, 2011.

[11] Ramnath Balasubramanyan and William W. Cohen. Block-lda: Jointly modeling entity-annotated text and entity-entity links. In *The 11th SIAM International Conference on Data Mining*, 2011.

[12] Ramnath Balasubramanyan, Frank Lin, and William W. Cohen. Node clustering in graphs: An empirical study. In *The NIPS 2010 Workshop on Networks Across Disciplines in Theory and Applications*, 2010.

[13] Shumeet Baluja, Rohan Seth, D. Sivakumar, Yushi Jing, Jay Yagnik, Shankar Kumar, Deepak Ravichandran, and Mohamed Aly. Video suggestion and discovery for YouTube: Taking random walks through the view graph. In *Proceeding of the 17th International Conference on World Wide Web*, 2008.

[14] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks. In *The 3rd International AAAI Conference on Weblogs and Social Media*, 2009.

[15] Richard Ernest Bellman. *Dynamic Programming*. Courier Dover Publications, 2003.

[16] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: Applications to image and text data. In *The 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001.

[17] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[18] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *The 24th Conference on Artificial Intelligence*, 2010.

[19] Andrew Carlson, Justin Betteridge, Richard C. Wang, Estevam R. Hruschka Jr., and Tom M. Mitchell. Coupled semi-supervised learning for information extraction. In *The 3rd ACM International Conference on Web Search and Data Mining*, 2010.

[20] Soumen Chakrabarti. Dynamic personalized PageRank in entity-relation graphs. In *The 16th International World Wide Web Conference*, 2007.

[21] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *The 34th Annual ACM Symposium on Theory of Computing*, 2002.

[22] Wen-Yen Chen, Yangqiu Song, Hongjie Bai, Chih-Jen Lin, and Edward Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[23] Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30, 2006.

[24] Anne Condon and Richard M. Karp. Algorithms for graph partitioning on the planted partition model. *Random Structures and Algorithms*, 18(2):116–140, 2001.

[25] Andrew Cotter, Joseph Keshet, and Nathan Srebro. Explicit approximations of the Gaussian kernel. *Pre-print*, 2011. arXiv:1109.4603v1 [cs.AI].

[26] Sanjoy Dasgupta. Experiments with random projection. In *The 16th Conference on Uncertainty in Artificial Intelligence*, 2000.

[27] J. de la Porte, B. M. Herbst, W. Hereman, and S. J. van der Walt. An introduction to diffusion maps. In *The 19th Symposium of the Pattern Recognition Association of South Africa*, 2008.

[28] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[29] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and L. Beck. Improving information retrieval using latent semantic indexing. *The 1988 Annual Meeting of the American Society for Information Science*, pages 36–40, 1988.

[30] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means, spectral clustering and normalized cuts. In *The 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[31] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007.

[32] Jacob Eisenstein, Brendan O'Connor, Noah A. Smith, and Eric P. Xing. A latent variable model for geographic lexical variation. In *The 2010 Conference on Empirical Methods in Natural Language Processing*, 2010.

[33] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *The Annual Conference of the Special Interest Group on Data Communication*, 1999.

[34] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[35] Xiaoli Zhang Fern and Carla E. Brodley. Random projection for high dimensional data clustering: A cluster ensemble approach. *Machine Learning*, 20(1):186–193, 2003.

[36] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[37] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7(2):179–188, 1936.

[38] Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the Nyström Method. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 26, pages 214–225, 2004.

[39] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28(2-3):133–168, 1997.

[40] Brian Gallagher, Hanghang Tong, Tina Eliassi-Rad, and Christos Faloutsos. Using ghost edges for classification in sparsely labeled networks. In *The 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

[41] Michelle Girvan and M. E. J. Newman. Community structure in social and biological networks. *The National Academy of Sciences of the United States of America*, 99(12):7821–7826, 2002.

[42] Zoltan Gyongyi, Hector Garcia-Molina, and Jan Pedersen. Web content categorization using link information. Technical report, Stanford University, 2006.

[43] Hui Han, Hongyuan Zha, and C. Lee Giles. Name disambiguation in author citations using a k-way spectral clustering method. In *The 2005 Joint Conference on Digital Libraries*, 2005.

[44] Taher Haveliwala, Sepandar Kamvar, and Glen Jeh. An analytical comparison of approaches to personalizing pagerank. Technical report, Stanford University, 2003.

[45] Jingrui He, Jaime Carbonell, and Yan Liu. Graph-based semi-supervised learning as a generative model. In *International Joint Conferences on Artificial Intelligence*, 2007.

[46] Robert Hecht-Nielsen. Context vectors: General purpose approximate meaning representations self-organized from raw data. *Computational Intelligence: Imitating Life*, pages 43–56, 1994.

[47] Yifen Huang. *Mixed-Initiative Clustering*. PhD thesis, Carnegie Mellon University, 2010.

[48] Alan J. Izenman. *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*. Springer, 2008.

[49] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mapping into Hilbert space. In *Conference in Modern Analysis and Probability*, 1984.

[50] Rosie Jones. *Learning to Extract Entities from Labeled and Unlabeled Text*. PhD thesis, Carnegie Mellon University, 2005.

[51] Anubhav Kale, Amit Karandikar, Pranam Kolari, Akshay Java, Tim Finin, and Anupam Joshi. Modeling trust and influence in the blogosphere using link polarity. In *The International Conference on Weblogs and Social Media*, 2007.

[52] Samuel Kaski. Dimensionality reduction by random mapping: fast similarity computation for clustering. In *The 1998 IEEE International Joint Conference on Neural Networks Proceedings*, 1998.

[53] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[54] Teuvo Kohonen. *Self-Organizing Maps*. Springer, 3rd edition, 2000.

[55] Zhenzhen Kou and William W. Cohen. Stacked graphical models for efficient inference in markov random fields. In *The 2007 SIAM International Conference on Data Mining*, 2007.

[56] Danai Koutra, Tai-You Ke, U Kang, Duen Horng (Polo) Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *The 2011 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2011.

[57] Valdis Krebs. unpublished. www.orgnet.com.

[58] Stéphane Lafon and Ann B. Lee. Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and data set parameterization. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 28(9):1393–1403, 2006.

[59] Ni Lao and William W. Cohen. Relational retrieval using a combination of path-constrained random walks. *Machine Learning*, 81(1):53–67, 2010.

[60] R.B. Lehoucq and D. C. Sorensen. Deflation techniques for an implicitly re-started arnoldi iteration. *SIAM Journal on Matrix Analysis and Applications*, 17:789–821, 1996.

[61] Jure Leskovec. *Dynamics of Large Networks*. PhD thesis, Carnegie Mellon University, 2008.

[62] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[63] Zhenguo Li, Jianzhuang Liu, Shifeng Chen, and Xiaoou Tang. Noise robust spectral clustering. In *IEEE 11th International Conference on Computer Vision*, 2007.

[64] Frank Lin and William W. Cohen. Power iteration clustering. In *The 27th International Conference on Machine Learning*, 2010.

[65] Frank Lin and William W. Cohen. Semi-supervised classification of network data using very few labels. In *The 2010 International Conference on Advances in Social Networks Analysis and Mining*, 2010.

[66] Frank Lin and William W. Cohen. A very fast method for clustering big text datasets. In *The 19th European Conference on Artificial Intelligence*, 2010.

[67] Frank Lin and William W. Cohen. Adaptation of graph-based semi-supervised methods to large-scale text data. In *The 9th Workshop on Mining and Learning with Graphs*, 2011.

[68] Wei Liu, Junfeng He, and Shih-Fu Chang. Large graph construction for scalable semi-supervised learning. In *The International Conference on Machine Learning*, 2009.

[69] Qing Lu and Lise Getoor. Link-based classification. In *The 20th International Conference on Machine Learning*, 2003.

[70] David Lusseau, Karsten Schneider, Oliver J. Boisseau, Patti Haase, and Elisabeth Slooten andSteve M. Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, 2003.

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

[72] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[73] Marina Meilă and Jianbo Shi. A random walks view of spectral segmentation. In *IEEE International Conference on Artificial Intelligence and Statistics*, 2001.

[74] Einat Minkov, Richard C. Wang, and William W. Cohen. Extracting personal names from email: Applying named entity recognition to informal text. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, 2005.

[75] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[76] Rajeev Motwani and Prabhaker Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[77] Miguel Á. Carreira-Perpi nán. Fast nonparametric clustering with Gaussian blurring mean-shift. In *Proceeds of the 23rd International Conference on Machine Learning*, 2006.

[78] Miguel Á. Carreira-Perpi nán. Generalised blurring mean-shift algorithms for nonparametric clustering. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[79] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(3):036104, 2006.

[80] Andrew Y. Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, 2002.

[81] Kamal Nigam and Rayid Ghani. Analyzing the effectiveness and applicability of co-training. In *The 9th International Conference on Information and Knowledge Management*, 2000.

[82] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[83] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814–818, 2005.

[84] Christos H. Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. In *The 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1998.

[85] Karl Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[86] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20*, 2007.

[87] Alan Ritter and Oren Etzioni. A latent Dirichlet allocation method for selectional preferences. In *The 48th Annual Meeting of the Association for Computational Linguistics*, 2010.

[88] Tom Roxborough and Arunabha Sen. Graph clustering using multiway ratio cut. In *Proceedings of Graph Drawing*, pages 291–296, 1997.

[89] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. A metric for distributions with applications to image databases. In *The 6th International Conference on Computer Vision*, 1998.

[90] Purnamrita Sarkar and Andrew W. Moore. Fast dynamic reranking in large graphs. In *The 18th International World Wide Web Conference*, 2009.

[91] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[92] Martin Szummer and Tommi Jaakkola. Partially labeled classification with Markov random walks. In *Advances in Neural Information Processing Systems 14*, 2001.

[93] Partha Pratim Talukdar, Joseph Reisinger, Marius Pasca, Deepak Ravichandran, Rahul Bhagat, and Fernando Pereira. Weakly-supervised acquisition of labeled class instances using graph random walks. In *The 2008 Conference on Empirical Methods in Natural Language Processing*, 2008.

[94] Lei Tang and Huan Liu. Scalable learning of collective behavior based on sparse social dimensions. In *The 18th ACM Conference on Information and Knowledge Management*, 2009.

[95] Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method. In *The 37th Annual Allerton Conference on Communication, Control and Computing*, 1999.

[96] Naftali Tishby and Noam Slonim. Data clustering by markovian relaxation and the information bottleneck method. In *Advances in Neural Information Processing Systems 13*, 2000.

[97] David Tolliver, Robert T. Collins, and Simon Baker. Multilevel spectral partitioning for efficient image segmentation and tracking. In *The 7th IEEE Workshops on Application of Computer Vision*, 2005.

[98] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Fast random walk with restart and its applications. In *The 2006 IEEE International Conference on Data Mining*, 2006.

[99] Benjamin Van Durme and Ashwin Lall. Online generation of locality sensitive hash signatures. In *The ACL 2010 Conference Short Papers*, 2010.

[100] Benjamin Van Durme and Ashwin Lall. Efficient online locality sensitive hashing via reservoir counting. In *The ACL 2011 Conference Short Papers*, 2011.

[101] Vladimir Naumovich Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[102] Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, March 1985.

[103] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[104] Xufei Wang, Lei Tang, Huiji Gao, and Huan Liu. Discovering overlapping groups in social media. In *The 2010 IEEE International Conference on Data Mining*, 2010.

[105] Tao Xiang and Shaogang Gong. Spectral clustering with eigenvector selection. *Pattern Recognition*, 41(3):1012–1029, 2008.

[106] Jian-Wu Xu, Puskal P. Pokharel, Kyu-Hwa Jeong, and Jose C. Principe. An explicit construction of a reproducing gaussian kernel Hilbert space. In *The 2006 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2006.

[107] Donghui Yan, Ling Huang, and Michael I. Jordan. Fast approximate spectral clustering. In *The 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009.

[108] Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.

[109] Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. In *Advances in Neural Information Processing Systems 17*, 2005.

[110] Kai Zhang, James T. Kwok, and Bahram Parvin. Prototype vector machine for large scale semi-supervised learning. In *The International Conference on Machine Learning*, 2009.

[111] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Scholkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems 16*, 2004.

[112] Hanson Zhou and David Woodruff. Clustering via matrix powering. In *The 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2004.

[113] Xiaojin Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2008.

[114] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *The 20th International Conference on Machine Learning*, 2003.

[115] Xiaojin Zhu, John Lafferty, and Zoubin Ghahramani. Combining active learning and semi-supervised learning using Gaussian fields and harmonic functions. In *ICML 2003 Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, 2003.

# Appendix A

# Evaluation Measures

## A.1 Classification

Here we define quantitative evaluation measures used for classification tasks according to a set of ground-truth labels. We assume a multi-class classification setting (i.e., the number of classes $k \geq 2$). All the measures used are in range $[0, 1]$ and in all the measures a higher value means better classification performance. Although most datasets used are single-label classification tasks (i.e., an instance in a dataset $\mathbf{x} \in \mathcal{X}$ has only one class label), with the exception of *accuracy*, the same evaluation measures are also used for multi-label classification tasks (i.e., $\mathbf{x}$ can have more than one class label).

Let $T_i$ be the set of instances with class label $i$ according to the ground-truth, $Y_i$ be the set of instances with class label $i$ according to the classifier output, and $n$ and $k$ be the number of instances and the number of clusters, respectively. Then:

**Accuracy (Acc)** is the fraction of correctly labeled instances in the entire dataset, in a single-label classification setting. Formally,

$$Acc = \frac{1}{n} \sum_i |T_i \cup Y_i|$$

**Precision (P)** is the fraction of predicted labels for a class that corresponds to the ground-truth. Formally, the precision of class $i$ is defined as

$$P_i = \frac{|T_i \cup Y_i|}{|Y_i|}$$

and the *macro-averaged precision* is the arithmetic mean of the precision of all the classes in the dataset:

$$P_M = \frac{1}{k} \sum_i P_i$$

**Recall (R)** is the fraction of ground-truth labels for a class that corresponds to the prediction. Formally, the recall of class $i$ is defined as

$$R_i = \frac{|T_i \cup Y_i|}{|T_i|}$$

and the *macro-averaged recall* is the arithmetic mean of the recall of all the classes in the dataset:

$$R_M = \frac{1}{k} \sum_i R_i$$

**F1-score (F1)** is the harmonic mean of precision and recall. Formally, the F1-score of class $i$ is defined as

$$F1_i = \frac{2P_iR_i}{P_i + R_i}$$

and the *macro-averaged F1-score* is the arithmetic mean of the F1-scores of all the classes in the dataset:

$$F1_M = \frac{1}{k} \sum_i F1_i$$

For additional background and details regarding these measures see Manning, Raghavan, and Schütze [72].

## A.2 Clustering

Here we define quantitative evaluation measures for clustering results according to a set of ground-truth labels. All the measures used are in range $[0, 1]$ and in all the measures a higher value means better classification performance. Unless otherwise noted, we assume a single-label setting where an instance in a dataset $\mathbf{x} \in \mathcal{X}$ can only belong to one cluster.

Let $T_i$ be the set of instances in the $i$-th ground-truth cluster, $C_j$ be the set of instances in the $j$-th predicted cluster, and $n = |\mathcal{X}|$ be the number of instances in the dataset. Then:

**Purity** is the best accuracy obtainable by a clustering, subject to the constraint that all elements in a cluster are assigned the same label. To compute purity, for each cluster $C_j$ returned by a clustering algorithm, the true labels of instances within $C_j$ are revealed and counted, and the entire cluster is assigned the label of with the highest count. Purity is then the accuracy of this assignment. Formally,

$$\text{Purity} = \frac{1}{n} \sum_j \max_i |T_i \cup C_j|$$

**Normalized mutual information (NMI)** is an information-theoretical measure where the mutual information of the ground-truth and the predicted clusters are nor-

malized by their respective entropies. Formally,

$$\text{NMI} = \frac{I(T, C)}{\frac{1}{2}(H(T) + H(C))}$$

where $I(T, C)$ is the mutual information between $T_1, T_2, \ldots$ and clustering $C_1, C_2, \ldots$ and $H(T)$ and $H(C)$ are their respective entropies, given by

$$I(T, C) = \sum_i \sum_j P(T_i, C_j) \log \frac{P(T_i, C_j)}{P(T_i)P(C_j)}$$

$$H(T) = -\sum_i P(T_i) \log P(T_i)$$

$$H(C) = -\sum_i P(C_i) \log P(C_i)$$

where $P(T_i) = \frac{|T_i|}{n}$ and $P(T_i, C_j) = \frac{|T_i \cup C_j}{n}$. Definitions for $P(C_j)$ follows.

**Rand index (RI)** compares the ground-truth and the predicted clusters for every possible pair of instances. The prediction for a particular pair is deemed correct if (a) they are in the same ground-truth cluster and in the same the predicted cluster, or (b) they are in different ground-truth clusters and in different predicted clusters. The Rand index is the number of correctly clustered pairs divided by the total number of possible pairs. Formally,

$$\text{RI} = \frac{2}{n(n-1)} \sum_{p,q|p<q} \begin{cases} 1 & \text{if } \exists i, j \mid \{x_p, x_q\} \in T_i \text{ and } \{x_p x_q\} \in C_j \\ 1 & \text{if } \nexists i \mid \{x_p, x_q\} \in T_i \text{ and } \nexists j \mid j\{x_p x_q\} \in C_j \\ 0 & \text{otherwise} \end{cases}$$

Note that RI is more expensive to calculate than other measures because it makes $O(n^2)$ comparisons.

In addition to the above methods, if we produce a mapping from ground-truth clusters $T_i$ to predicted clusters $C_j$, then classification evaluation measures such as the ones

found in Appendix A.1 can be used to evaluate the mapped clusters. In this work we use the Hungarian algorithm whenever we use classification measures such as precision, recall, or F1-score for cluster evaluation. For additional background and details regarding the above mentioned measures see Manning, Raghavan, and Schütze [72].

# Appendix B

# Additional Experiment Results

## B.1 SSL Methods on Network Datasets

For comparing difference between HF and MRW when using CountLink and PageRank seed preferences, a one-tail paired McNemar's test on classification results of individual instances is used with $p < 0.001$ reported as significant. For comparing difference between HF and MRW when using Random seed preference, 20 accuracy scores from 20 random trials are used in a one-tail Mann-Whitney U test with $p < 0.001$ reported as significant. For comparison difference random seeding and authority-based seed preferences, classification results of individual instances is used in a one-tail Mann-Whitney U test with $p < 0.05$ reported as significant.

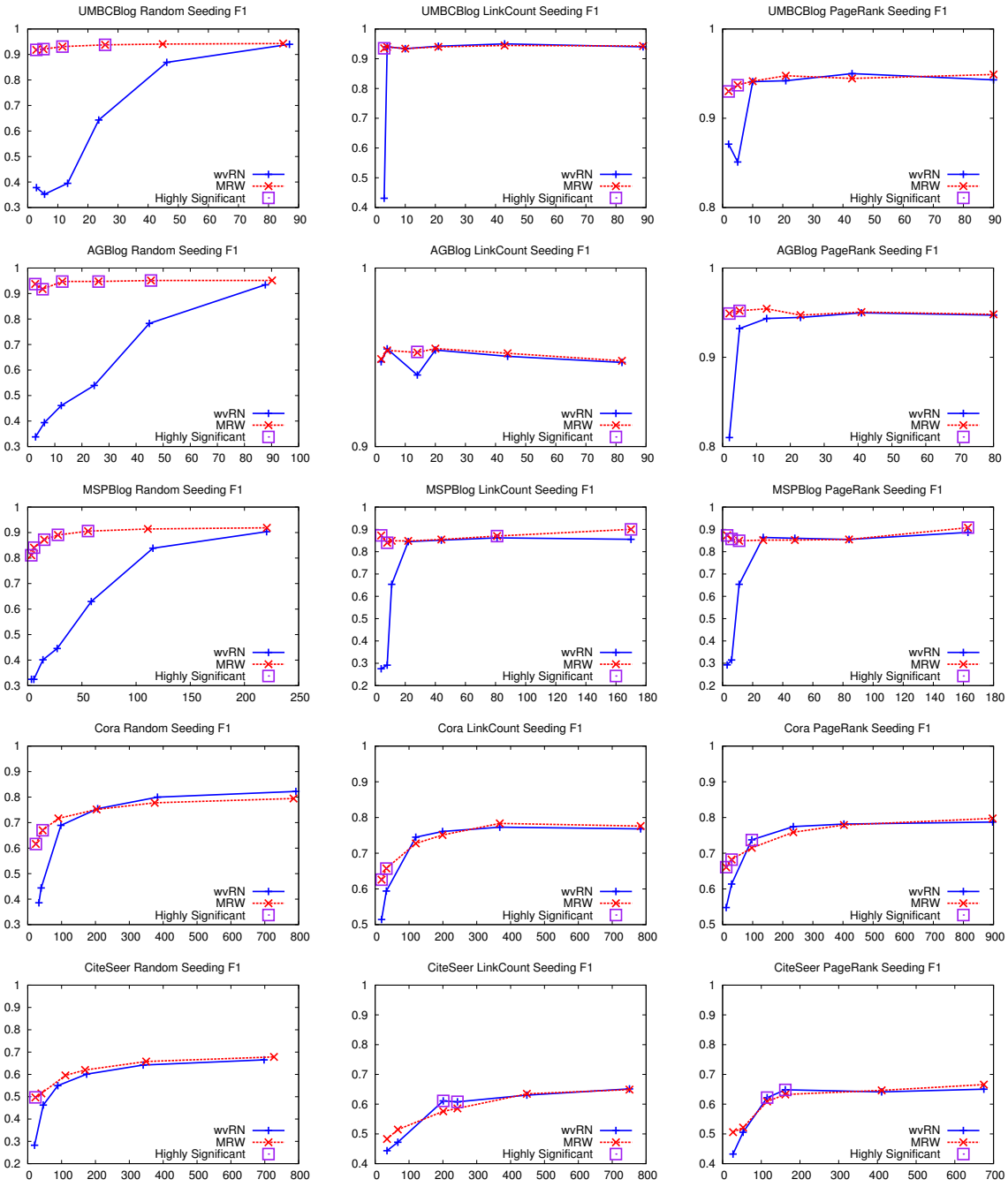



Figure B.1: F1 score results varying the learning algorithm. The x-axis indicates number of labeled instances and y-axis indicates labeling macro-averaged F1 score. Square block around a point indicates statistical significance with $p < 0.001$.

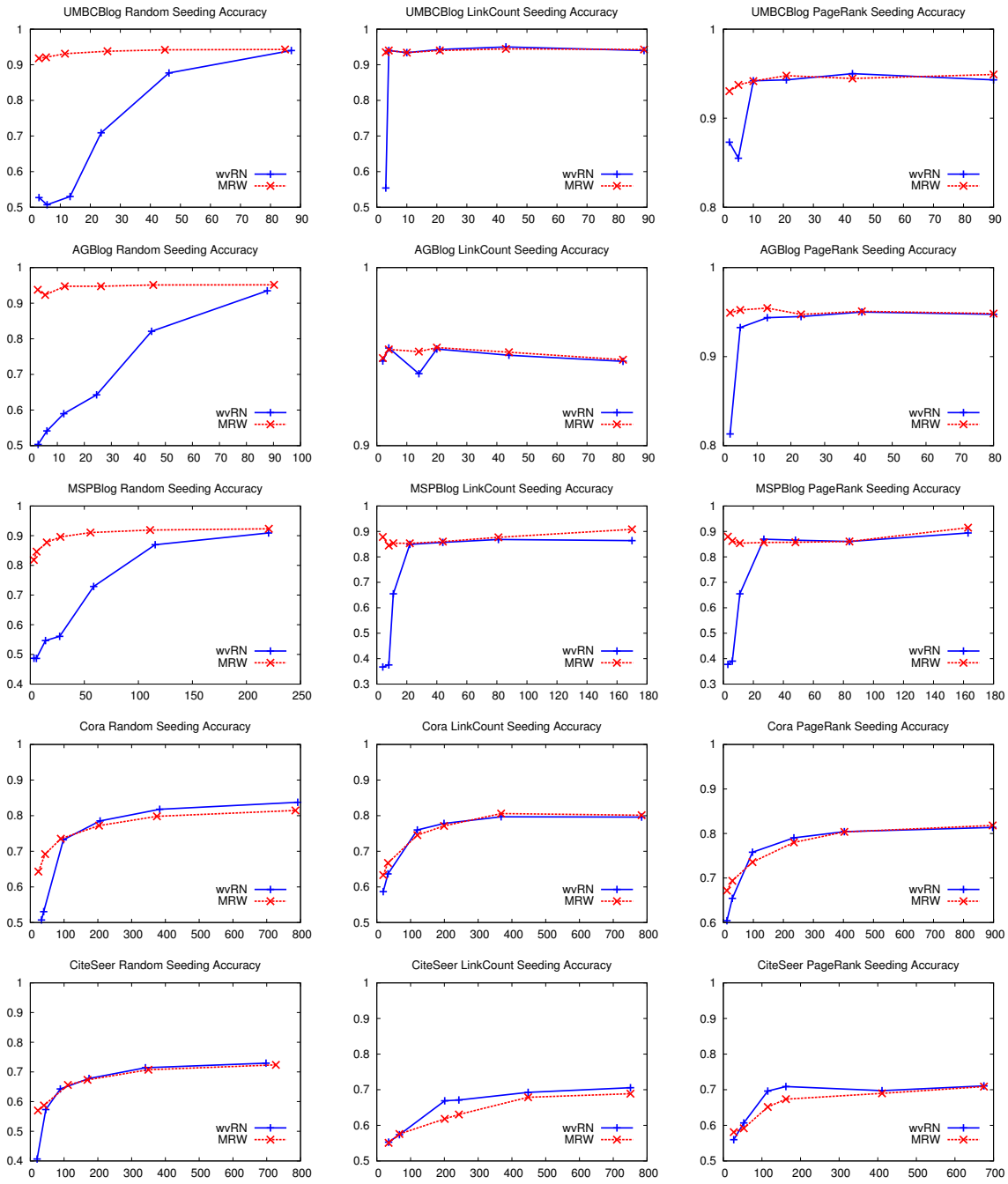



Figure B.2: Accuracy results varying the learning algorithm. The x-axis indicates number of labeled instances and y-axis indicates labeling accuracy.

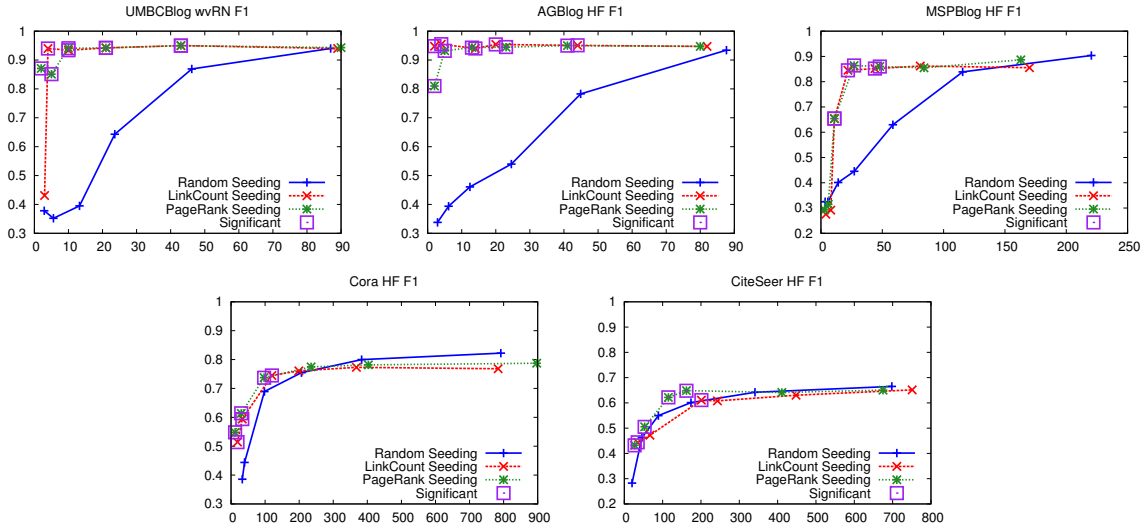



Figure B.3: Harmonic fields F1 score results varying the seeding method. The x-axis indicates number of labeled instances and y-axis indicates labeling macro-averaged F1 score. Square block around a point indicates statistical significance with $p < 0.05$.

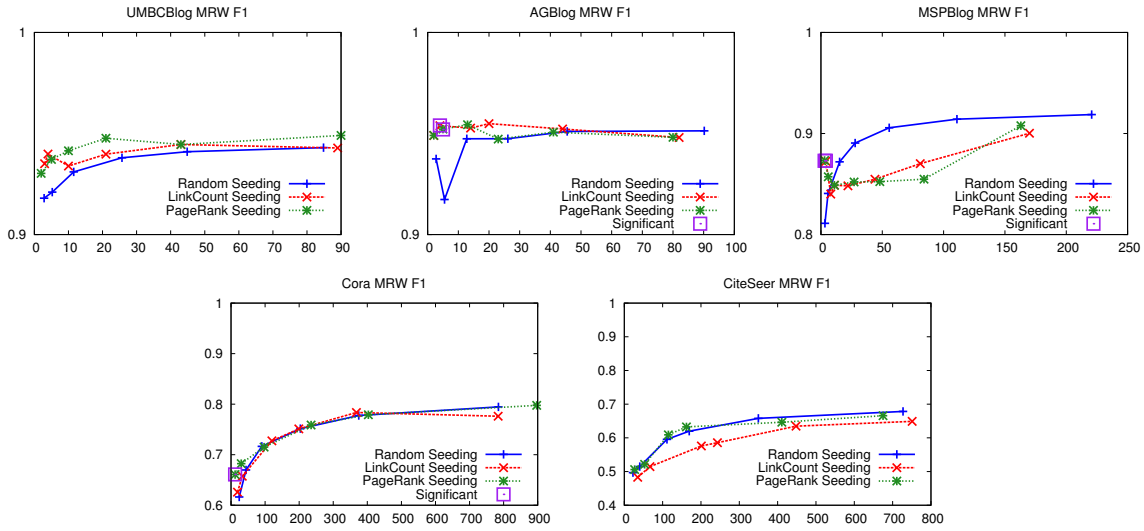



Figure B.4: MultiRankWalk F1 score results varying the seeding method. The x-axis indicates number of labeled instances and y-axis indicates labeling macro-averaged F1 score. Square block around a point indicates statistical significance with $p < 0.05$.

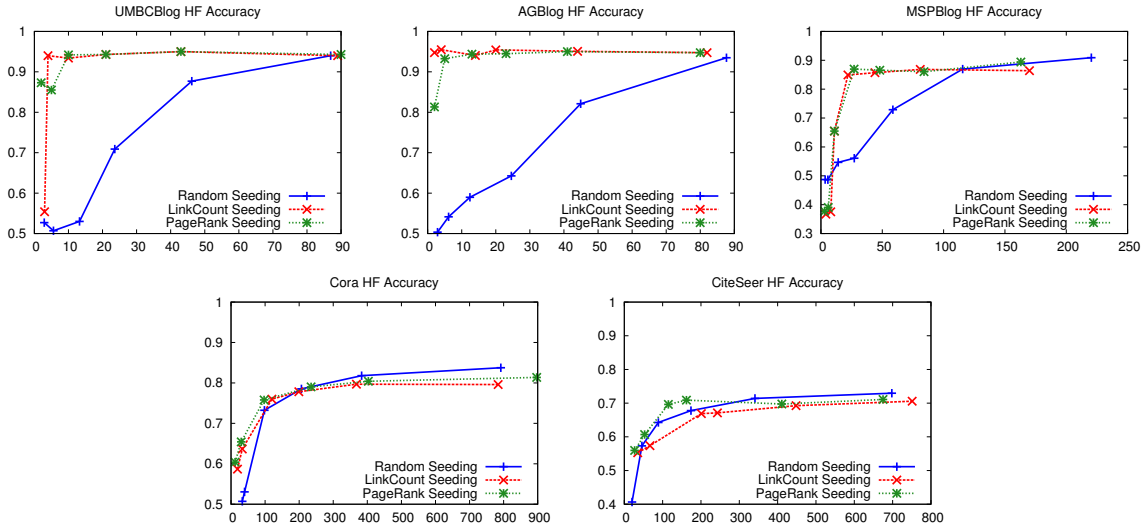



Figure B.5: Harmonic fields accuracy results varying the seeding method. The x-axis indicates number of labeled instances and y-axis indicates labeling accuracy.

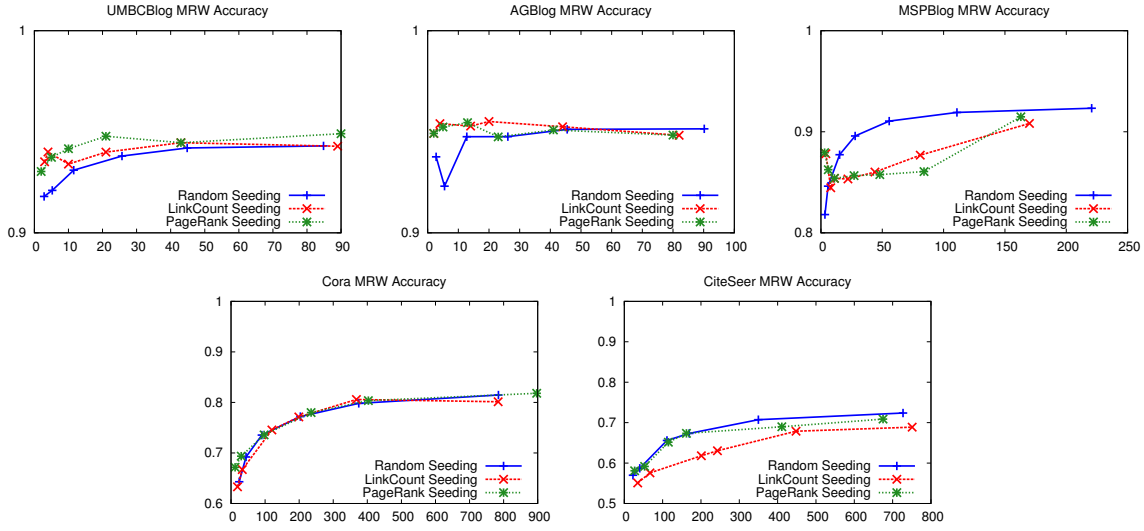



Figure B.6: MultiRankWalk accuracy results varying the seeding method. The x-axis indicates number of labeled instances and y-axis indicates labeling accuracy.

## B.2 SSL and Clustering Methods on Network Datasets

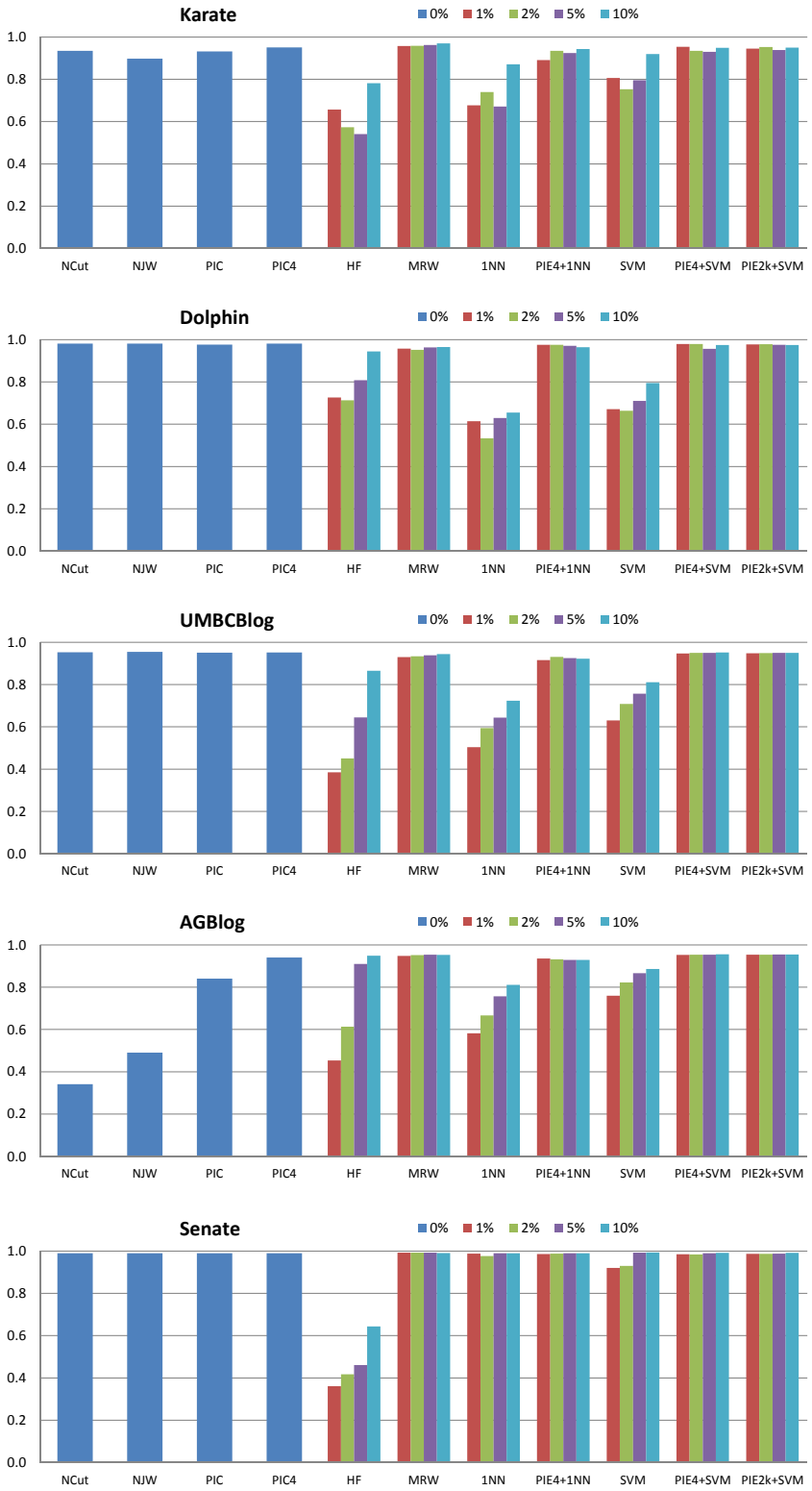



Figure B.7: F1 comparison between SSL methods and clustering methods on network data.

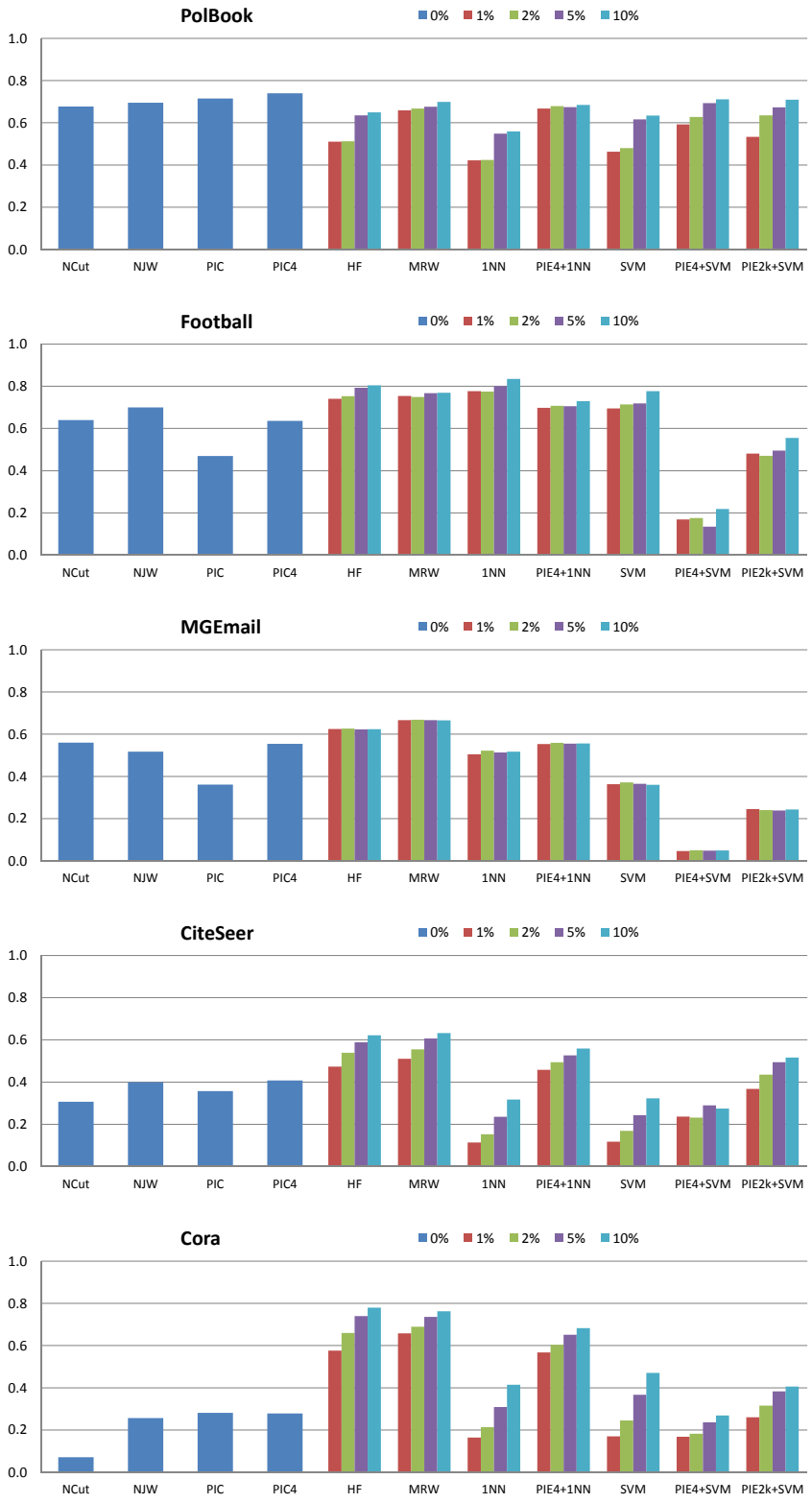
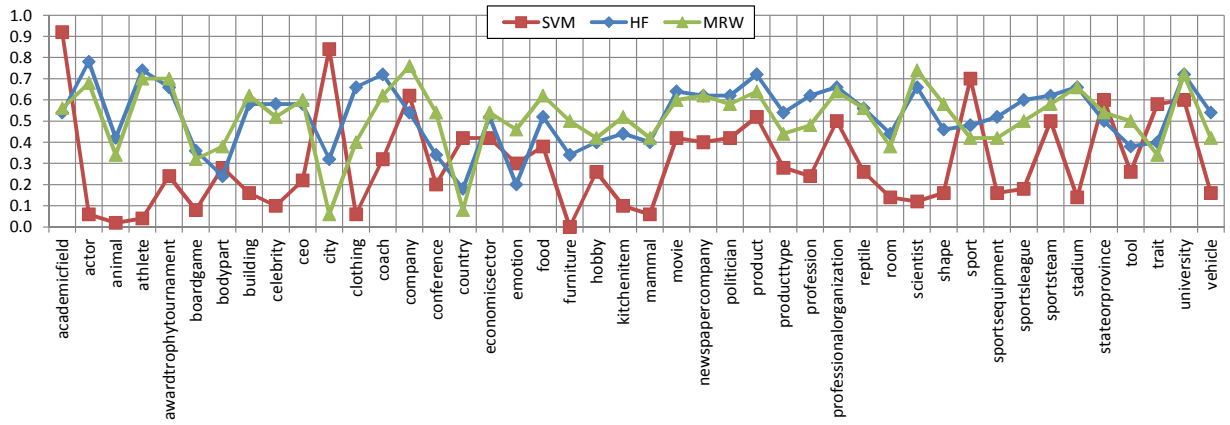
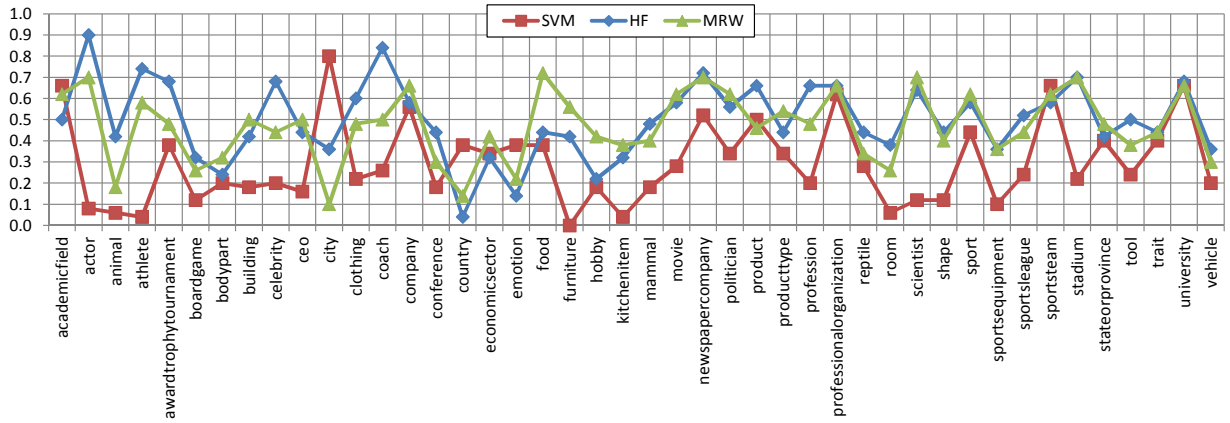



Figure B.8: F1 comparison between SSL methods and clustering methods on network data.

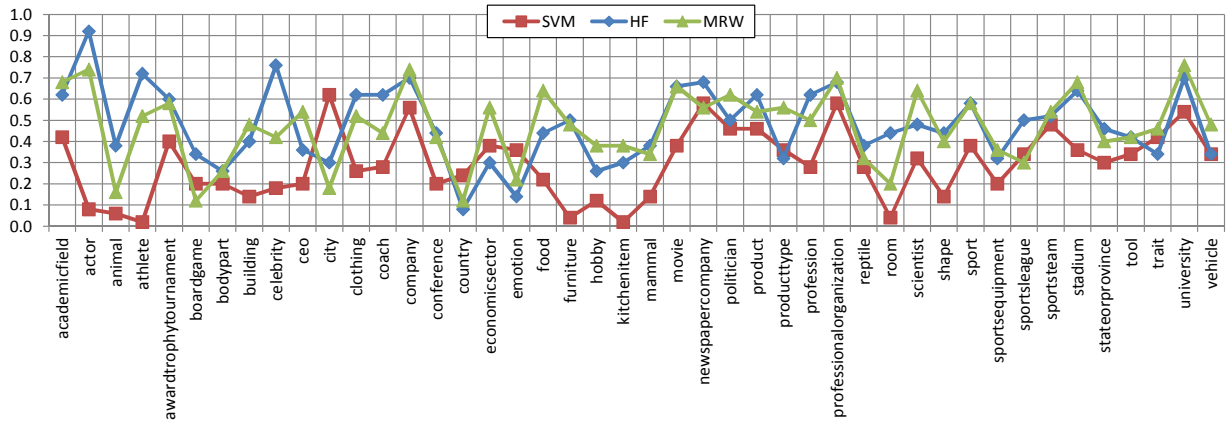
## B.3 IM SSL Methods on 44Cat




(a) Sampled per-category accuracy of the top 100 NPs




(b) Sampled per-category accuracy of the top 500 NPs




(c) Sampled per-category accuracy of the top 1000 NPs

Figure B.9: Sampled per-category accuracies of the top NPs on the 44Cat dataset. The categories are ordered from left to right according to the difference between the MRW accuracy and HF accuracy, from the high to low.

# Appendix C

# Relation between HF and co-EM

The basic idea behind co-EM is to combine features of co-training (having two views of the data) and Expectation-Maximization (iteratively maximize data likelihood) as an iterative bootstrapping classifier. The bipartite graph walk can be thought of as having two views with two types of classifiers: first, we train a feature classifier based on the instance labels and classifies (walks to) the features; then we train an instance classifier based on the newly labeled features and classifies (walks to) the instances. Co-EM with these two views and two classifiers proved to be effective in extracting noun phrases in [50].

Algorithm 12 is the co-EM algorithm for extraction of noun phrases [50], where $\hat{f}_n$ and $\hat{f}_c$ are the two classifiers corresponding to two views of the data, noun phrases and contexts, respectively. If we let $\boldsymbol{v}^t$ be the vector corresponding to $\hat{f}_n$ and $F$ be the matrix form of co-occurrence data $X$, then step 6 of Algorithm 12 can be computed by $C^{-1}F\boldsymbol{v}^t$ (where $C$ is the diagonal matrix of column sums), and thus step 8 can be computed directly by $R^{-1}F(C^{-1}F\boldsymbol{v}^t)$ (where $R$ is the diagonal matrix of row sums), which is equivalent to a single-class version of Equation 6.7. This shows that co-EM can be viewed as a graph-based SSL method with the implicit manifold constructed according to the particular classifiers and two views of the data. While this connection is based

on a particular formulation of co-EM, it generalizes to any classifiers/views where the computation can be described in terms of sparse matrix operations.

**Algorithm 12** The co-EM algorithm for information extraction [50]

**procedure** coEM($X, \mathcal{L}$)
  **if** $X_{i:} \in \mathcal{L}$ **then** $\hat{f}_{n0}(X_{i:}) = 1$
  **else** $\hat{f}_{n0}(X_{i:}) = 0$
  **end if**
  **repeat**
    $\hat{f}_c(X_{:j}) = \frac{\sum_{X_{i:}} \hat{f}_n(X_{i:}) * X_{ij}}{\sum_i X_{ij}}$
    **if** $X_{i:} \in \mathcal{L}$ **then** $\hat{f}_n(X_{i:}) = 1$
    **else** $\hat{f}_n(X_{i:}) = \frac{\sum_{X_{:j}} \hat{f}_c(X_{:j}) * X_{ij}}{\sum_j X_{ij}}$
    **end if**
  **until** $\hat{f}_n$ has converged **return** $\hat{f}_n, \hat{f}_c$
**end procedure**

# Appendix D

# Math

## D.1 Power Series of Damped Column-Stochastic Matrix

We want to show the following equation holds:

$$\lim_{t\to\infty} \sum_{i=0}^{t-1} (dP)^i = (I - dP)^{-1} \tag{D.1}$$

Let $M = dP$. Then

$$\begin{aligned}
\lim_{t\to\infty} \sum_{i=0}^{t-1} (dP)^i &= \lim_{t\to\infty} \sum_{i=0}^{t-1} M^i \\
&= \lim_{t\to\infty} (M^0 + M^1 + M^2 + ... + M^{t-1}) \\
&= \lim_{t\to\infty} (I + M^1 + M^2 + ... + M^{t-1})
\end{aligned}$$

Multiply by $(I-M)^{-1}$:

$$\begin{aligned}\left(\lim_{t\to\infty}\sum_{i=0}^{t-1}M^i\right)(I-M) &= \left(\lim_{t\to\infty}(I+M^1+M^2+...+M^{t-1})\right)(I-M)\\ &= \lim_{t\to\infty}\left((I+M^1+M^2+...+M^{t-1})(I-M)\right)\\ &= \lim_{t\to\infty}\left((I-M)+(M-M^2)+(M^2-M^3)+...+(M^{t-1}-M^t)\right)\\ &= \lim_{t\to\infty}(I-M^t)\end{aligned}$$

Since $0 < d < 1$ and the eigenvalues of $P$ are in $[-1,1]$, $\lim_{t\to\infty} M^t = 0$, therefore we have

$$\left(\lim_{t\to\infty}\sum_{i=0}^{t-1}M^i\right)(I-M) = I$$

Finally,

$$\left(\lim_{t\to\infty}\sum_{i=0}^{t-1}M^i\right)(I-M) = (I-M)^{-1}(I-M)$$

$$\left(\lim_{t\to\infty}\sum_{i=0}^{t-1}(dP)^i\right)(I-dP) = (I-dP)^{-1}(I-dP)$$

$$\lim_{t\to\infty}\sum_{i=0}^{t-1}(dP)^i = (I-dP)^{-1}$$

□

## D.2 GKHS Approximation with Taylor Features Detail

We can rewrite the Gaussian kernel from Equation 8.1 as:

$$\begin{aligned} K(\mathbf{x}, \mathbf{y}) &= e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}} \\ &= e^{-\frac{(\mathbf{x}-\mathbf{y})^T(\mathbf{x}-\mathbf{y})}{2\sigma^2}} \\ &= e^{-\frac{\mathbf{x}^T\mathbf{x}-2\langle \mathbf{x},\mathbf{y}\rangle+\mathbf{y}^T\mathbf{y}}{2\sigma^2}} \\ &= e^{-\frac{\|\mathbf{x}\|^2}{2\sigma^2}} \cdot e^{-\frac{\|\mathbf{y}\|^2}{2\sigma^2}} \cdot e^{\frac{\langle \mathbf{x},\mathbf{y}\rangle}{\sigma^2}} \end{aligned} \tag{D.2}$$

The construction for the first and second terms are straight forward, so we will focus on the third term. Using Taylor expansion of the exponential function about 0 we have:

$$e^{\frac{\langle \mathbf{x},\mathbf{y}\rangle}{\sigma^2}} = \sum_{k=0}^{\infty} \frac{1}{k!} \left( \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\sigma^2} \right)^k \tag{D.3}$$

Expanding $\langle \mathbf{x}, \mathbf{y} \rangle^k$ we get:

$$\langle \mathbf{x}, \mathbf{y} \rangle^k = \left( \sum_{i=1}^{m} \mathbf{x}_i \mathbf{y}_i \right)^k = \sum_{j \in [m]^k} \left( \prod_{i=1}^{k} \mathbf{x}_{j_i} \right) \left( \prod_{i=1}^{k} \mathbf{y}_{i_j} \right)$$

where $j$ enumerates all possible selections of $k$ elements of $\mathbf{x}$ or $\mathbf{y}$ as per definition of raising a sum of $m$ terms to the $k$th power. Plugging this back into Equation D.3 we get:

$$\begin{aligned} \sum_{k=0}^{\infty} \frac{1}{k!} \left( \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\sigma^2} \right)^k &= \sum_{k=0}^{\infty} \frac{1}{k!} \left( \frac{\sum_{j \in [m]^k} \left( \prod_{i=1}^{k} \mathbf{x}_{j_i} \right) \left( \prod_{i=1}^{k} \mathbf{y}_{i_j} \right)}{\sigma^{2k}} \right) \\ &= \sum_{k=0}^{\infty} \frac{1}{\sigma^{2k} k!} \sum_{j \in [m]^k} \left( \prod_{i=1}^{k} \mathbf{x}_{j_i} \right) \left( \prod_{i=1}^{k} \mathbf{y}_{i_j} \right) \end{aligned}$$

and plugging this back into Equation D.2 we get:

$$
\begin{aligned}
e^{-\frac{\|x-y\|^2}{2\sigma^2}} &= e^{-\frac{\|x\|^2}{2\sigma^2}} \cdot e^{-\frac{\|y\|^2}{2\sigma^2}} \cdot \sum_{k=0}^{\infty} \frac{1}{\sigma^{2k} k!} \sum_{j \in [m]^k} \left( \prod_{i=1}^{k} x_{j_i} \right) \left( \prod_{i=1}^{k} y_{i_j} \right) \\
&= \sum_{k=0}^{\infty} \frac{1}{\sigma^{2k} k!} \sum_{j \in [m]^k} e^{-\frac{\|x\|^2}{2\sigma^2}} \left( \prod_{i=1}^{k} x_{j_i} \right) e^{-\frac{\|y\|^2}{2\sigma^2}} \left( \prod_{i=1}^{k} y_{i_j} \right) \\
&= \sum_{k=0}^{\infty} \sum_{j \in [m]^k} e^{-\frac{\|x\|^2}{2\sigma^2}} \frac{1}{\sigma^k \sqrt{k!}} \left( \prod_{i=1}^{k} x_{j_i} \right) e^{-\frac{\|y\|^2}{2\sigma^2}} \frac{1}{\sigma^k \sqrt{k!}} \left( \prod_{i=1}^{k} y_{i_j} \right)
\end{aligned}
$$

Therefore an explicit feature representation $\phi$ for each $k$ and $j$ is:

$$
\phi_{k,j}(x) = e^{-\frac{\|x\|^2}{2\sigma^2}} \frac{1}{\sigma^k \sqrt{k!}} \prod_{i=1}^{k} x_{j_i}
$$

where $K(x, y) = \langle \phi(x), \phi(y) \rangle = \sum_{k=0}^{\infty} \sum_{j \in [m]^k} \phi_{k,j}(x) \phi_{k,j}(y)$, where $[m]^k$ denotes the set of permutations of size $k$ on $m$ objects. An obvious problem with putting this formulation into practice is the summing up of infinite terms indexed by $k$; therefore we can approximate it by restricting $k \leq d$–thus the Gaussian kernel is approximated by polynomials of degree $d$ via a truncated Taylor expansion:

$$
\hat{K}(x, y) = \sum_{k=0}^{d} \sum_{j \in [m]^k} \phi_{k,j}(x) \phi_{k,j}(y) \tag{D.4}
$$

## D.3 Taylor Features about Non-zero Points

Equation 8.3 uses Taylor expansion of the exponential function about 0. If we generalize the expansion about any real number $a$, in place of Equation 8.3 we have:

$$
e^{\frac{\langle x, y \rangle}{\sigma^2}} = \sum_{k=0}^{\infty} \frac{e^a}{k!} \left( \frac{\langle x, y \rangle}{\sigma^2} - a \right)^k = \sum_{k=0}^{\infty} \frac{e^a}{k!} \left( \frac{\langle x, y \rangle - \sigma^2 a}{\sigma^2} \right)^k \tag{D.5}
$$

Expanding $(\langle x, y\rangle - \sigma^2 a)^k$ we get:

$$(\langle x, y\rangle - \sigma^2 a)^k = \left(\sum_{i=1}^{m} x_i y_i - \sigma^2 a\right)^k = \sum_{j\in[m']^k} \left(\prod_{i=1}^{k} x'_{j_i}\right)\left(\prod_{i=1}^{k} y'_{i_j}\right)$$

where $m' = m + 1$, and $x'$ and $y'$ are $x$ and $y$ appended with an additional constant coordinate $i\sigma\sqrt{a}$ (with $i$ being the imaginary number). Now $j$ enumerates all possible selections of $k$ elements from $m$ coordinates of $x$ and $y$ plus $i\sigma\sqrt{a}$. Note that the imaginary number $i$ will always be canceled out during computation. Plugging this back into Equation D.5 we have:

$$\begin{aligned}\sum_{k=0}^{\infty} \frac{e^a}{k!}\left(\frac{\langle x, y\rangle - \sigma^2 a}{\sigma^2}\right)^k &= \sum_{k=0}^{\infty} \frac{e^a}{k!}\left(\frac{\sum_{j\in[m']^k}\left(\prod_{i=1}^{k} x'_{j_i}\right)\left(\prod_{i=1}^{k} y'_{i_j}\right)}{\sigma^{2k}}\right)\\ &= \sum_{k=0}^{\infty} \frac{e^a}{\sigma^{2k}k!}\sum_{j\in[m']^k}\left(\prod_{i=1}^{k} x'_{j_i}\right)\left(\prod_{i=1}^{k} y'_{i_j}\right)\end{aligned}$$

and plugging this into Equation 8.2 we get:

$$\begin{aligned}e^{-\frac{\|x-y\|^2}{2\sigma^2}} &= e^{-\frac{\|x\|^2}{2\sigma^2}}\cdot e^{-\frac{\|y\|^2}{2\sigma^2}}\cdot\sum_{k=0}^{\infty}\frac{e^a}{\sigma^{2k}k!}\sum_{j\in[m']^k}\left(\prod_{i=1}^{k} x'_{j_i}\right)\left(\prod_{i=1}^{k} y'_{i_j}\right)\\ &= \sum_{k=0}^{\infty}\frac{e^a}{\sigma^{2k}k!}\sum_{j\in[m']^k} e^{-\frac{\|x\|^2}{2\sigma^2}}\left(\prod_{i=1}^{k} x'_{j_i}\right) e^{-\frac{\|y\|^2}{2\sigma^2}}\left(\prod_{i=1}^{k} y'_{i_j}\right)\\ &= \sum_{k=0}^{\infty}\sum_{j\in[m']^k} e^{-\frac{\|x\|^2}{2\sigma^2}}\frac{e^{\frac{a}{2}}}{\sigma^k\sqrt{k!}}\left(\prod_{i=1}^{k} x'_{j_i}\right) e^{-\frac{\|y\|^2}{2\sigma^2}}\frac{e^{\frac{a}{2}}}{\sigma^k\sqrt{k!}}\left(\prod_{i=1}^{k} y'_{i_j}\right)\end{aligned}$$

Therefore a generalized explicit feature representation $\phi$ for each $k$ and $j$ about $a$ is:

$$\phi_{k,j}(x) = e^{-\frac{\|x\|^2}{2\sigma^2}}\frac{e^{\frac{a}{2}}}{\sigma^k\sqrt{k!}}\prod_{i=1}^{k} x'_{j_i} = e^{\frac{\sigma^2 a - \|x\|^2}{2\sigma^2}}\frac{1}{\sigma^k\sqrt{k!}}\prod_{i=1}^{k} x'_{j_i} \tag{D.6}$$

where $K(\boldsymbol{x}, \boldsymbol{y}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle = \sum_{k=0}^{\infty} \sum_{j \in [m']^k} \phi_{k,j}(\boldsymbol{x}') \phi_{k,j}(\boldsymbol{y}')$, where $[m']^k$ denotes the set of permutations of size $k$ on $m+1$ objects. Note Equation D.6 is very similar to 8.4.