Adaptation of Graph-Based Semi-Supervised Methods to Large-Scale Text Data

Frank Lin
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
frank@cs.cmu.edu

William W. Cohen
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
wcohen@cs.cmu.edu

ABSTRACT

Graph-based semi-supervised learning methods have shown to be efficient and effective on network data by propagating labels along neighboring nodes. These methods can also be applied to general data by constructing a graph where the nodes are the instances and the edges are weighted by the similarity between feature vectors of instances. However, whereas a natural network is often sparse, a network of pairwise similarities between instances is dense, and prohibitively large for even moderately sized text datasets. We show, through using a simple general technique, how these learning methods can be *exactly and efficiently* applied to text data—using the complete pair-wise similarity manifold—without resorting to sampling or sparsification. This technique also provides a unifying view of prior work on label propagation on text graphs, and we assess its effectiveness applied to two popular graph-based semi-supervised methods on several large real datasets.

1. INTRODUCTION

Traditional supervised learning methods learns from labeled instances, and how well they learn depend on the amount of labeled data available. Labels often require substantial human effort, giving rise to *semi-supervised learning* (SSL), which learns from both labeled data (expensive to obtain) and unlabeled data (of which there is usually plenty), often by leveraging the similarity between data points [26].

Graph-based SSL methods are known for their simplicity, effectiveness, and scalability [27, 16, 20]; in particular, they are well-suited for large datasets with "natural" graph structures, such as a YouTube video dataset where nodes represent videos and weighted edges between nodes represent how often the same user view these two videos [3]. These methods are efficient for large datasets because (1) they are methods that propagate labels from labeled to unlabeled nodes through the edges and (2) most large graphs are sparse in the number of edges.

However, data from natural language tasks often do not come naturally in the form of a graph. Here we consider text classification and information extraction by noun phrase classification; to use many of these SSL methods, we first need to construct a manifold in the form of a pair-wise similarity matrix—a graph where the nodes are data points and weighted edges denote pair-wise similarity between points. This manifold is a powerful and useful representation, allowing graph-based SSL methods to be applied to any dataset given a similarity measure. But there is a caveat—this graph is almost always dense ($\sim n^2$ edges).

With dense graphs many methods are no longer scalable. To construct, to store, and to operate on such a graph is $O(n^2)$ in terms of both time and space. In making SSL learning methods practical on general dense graph data, prior work has mostly relied on *sparsifying techniques* by either sampling nodes or edges [24, 6] or constructing a new, smaller graph that is representative of the original graph [15]. The construction of these sparsified graphs often require some expertise and familiarity with the technique and incur additional non-trivial computation costs (e.g., constructing a k-nearest neighbor graph using a kd-tree, inverted index, or locality sensitive hashing). In contrast, we propose a simple yet effective solution to this problem that is *exactly equivalent to using a* **complete** *pair-wise similarity manifold* while keeping runtime and storage linear to the input size, without sampling or calculating a sparse representation of the data. This is possible using a "path-folding trick", which is based on a simple observation: many iterative semi-supervised learning methods have at their core matrix-vector multiplications, where the matrix is based on the adjacency matrix of the graph. If the graph is sparse (number of edges $|E| = O(n)$), these methods terminate quickly and require small amounts of memory. If the graph is dense (e.g., $|E| = O(n^2)$), the methods become slow and require large amounts of memory. Thus, if we are able to decompose a dense similarity 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. We call this *implicit manifold construction*.

While prior work in text mining has explored learning through label propagation on graphs constructed from text data [11, 22, 5, 4], it is often not clear how these methods are related each other or how they are related to SSL method outside text mining literature. By looking at each of these methods as one of two basic SSL label propagation methods with implicit manifold construction using different manifolds, seemingly different propagation algorithm may be brought under a unifying view, thus creating an organized, general set of tools which NLP researchers can utilize methodically for large-scale text mining.

Additional contributions of this work are: (1) implicit manifold versions of two widely used graph-based SSL methods, (2) alternate manifolds for these methods based on different similarity functions (inner product, cosine, and bipartite graph walk), and (3) evaluation and comparison of these two methods on a number of large, non-network datasets using implicit manifolds.

The rest of this paper examines this solution in more detail. First we present "path-folding" and how it can be used for scalable SSL (Section 2). Next we look at two popular graph-based SSL methods (Section 2.1), why they are natural candidates for implicit manifolds, and how we can "plug in" different similarity functions resulting in different manifolds (Section 2.2). Then we study the effectiveness and behavior of these methods on four medium to large datasets (Section 3) and conclude that graph-based SSL methods can be efficiently and effectively applied to large non-graph data via implicit manifold construction (Section 4).

2. SSL WITH IMPLICIT MANIFOLDS

The idea of path-folding (PF) is often used in network science to transform a two-mode network (graph with two distinct types of nodes) into a one-mode network, and it has been used as an efficient way to apply a graph-based clustering method to large text datasets [14]. Here we extend the application to graph-based SSL methods and general datasets. 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 its 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 would 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:

$$FF^T = S \quad (1)$$

where rows of F represent instances and columns of F represent features. S is then the "folded" graph—each node is an instance, and a weighted edge between two instances ($S(i, j)$) represent 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 S, as the size of the dataset (the number of instances) grows. In real datasets, F can often be consider 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). S, on the other hand, is likely dense. As examples, for census data S will be a full matrix (a complete graph), and for document data $S(i, j)$ is zero only if no words are shared between documents i and j—yet the frequent occurrence of many common words makes $S(i, j) = 0$ highly unlikely.

As the number of instances increases, S, a direct representation of a similarity manifold, becomes very costly to store and operate on. Its decomposition FF^T, on the other hand, is a much more compact representation. This observation—the equivalence of S and FF^T together with the contrast between their density—becomes a powerful tool when applied to certain algorithms. Specifically, any algorithm involving the multiplication of the similarity matrix S with a vector can *turn a dense matrix-vector multiplication into a series of sparse matrix-vector multiplications*: instead of $v' = Sv$, we do $v' = F(F^T v)$ (note the parentheses), thus turning an $O(n^2)$ operation into $O(n)$ while getting the exact same result. Next, we look at two representative graph-based SSL methods and how we can apply path-folding to them and adapt them to different types of data.

2.1 Two Graph-based SSL Methods

We denote the input data as X, composed of unlabeled instances X^U and labeled instances X^L with corresponding labels Y^L. Since in this section the input is a graph with edges representing similarity, X is equivalent to the similarity matrix S. We also define $D = \sum_i S_i$, and thus $D^{-1}S$ is the row-normalized similarity matrix and SD^{-1} the column-normalized one.

The Harmonic Functions Method

The harmonic functions method (HF), proposed in [27], is a popular graph-based label propagation SSL method that assigns labels based the harmonic function values induced over the graph given some training instances. HF assumes homophily between nodes in the graph, and the harmonic function value assigned to a node is also the probability of a random walk from that node hitting a positively labeled node before hitting a negative one (generalizes to multi-class cases). It is one of the best classifiers on many benchmark network datasets as noted in [16], where it is referred to as wvRN. HF for large sparse graphs can be solved simply and efficiently using an iterative algorithm:

Algorithm 1 An iterative harmonic functions alg.

```
Input: $S = \{X^L, X^U\}, Y^L$
Output: Labels $Y^U$ for unlabeled instances $X^U$
  1. $V^0_{ci} \leftarrow 1$ if $Y^L_i = c$, else $V_{ci} \leftarrow 0$ and $t \leftarrow 0$
  2. $V^{t+1} \leftarrow D^{-1}SV^t$;
  3. $\forall i \in Y^L : V^{t+1}_i \leftarrow V^0_i$ and $t \leftarrow t + 1$
  4. Go to step 2 unless $V^t$ has converged
  5. $Y^U_i \leftarrow argmax_c(V_{ci})$
```

MultiRankWalk

MultiRankWalk (MRW) [13] is a graph-based SSL method based on random graph walk (RW). It is directly related to personalized PageRank [9] and random walk with restart [23]. Given a graph represented by matrix S, a vector of random walk probability distribution over the nodes $\mathbf{v}$ is satisfies the equation $\mathbf{v} = \alpha\mathbf{r} + (1 - \alpha)SD^{-1}\mathbf{v}$, where SD^{-1} is the column-stochastic transition matrix of the graph, $\mathbf{r}$ is a normalized restart vector where $||\mathbf{r}||_1 = 1$, and α is the *restart probability*. The vector $\mathbf{v}$ can be interpreted as the probability distribution of a random walk on S, but at each step there is an α probability of "teleporting" to a random node with distribution $\mathbf{r}$. The basic idea of SSL via random walks is to (1) do multiple RW's, one for each class c, with the labeled instances from c defining the restart vector; then (2) assign label to an instance i by comparing its scores across different RW distributions. See Algorithm 2 for a basic MRW algorithm.

HF vs. MRW

Many of the recent graph-based label-propagation methods can be viewed as a variation of HF [16, 3, 22, 20] or MRW [21, 25, 8, 10, 7], , and while both assume homophily in data and are related to random walks, they are different propagation methods and produce very different results. Viewed in light of random graph walks,

Algorithm 2 A basic MultiRankWalk algorithm

```
Input: $S = \{X^L, X^U\}, Y^L$
Output: Labels $Y^U$ for unlabeled instances $X^U$
  1. $R_{ci} \leftarrow 1$ if $Y_i^L = c$, else $R_{ci} \leftarrow 0$
  2. Norm. columns of $R$ to sum to 1
  3. $V^0 \leftarrow R$ and $t \leftarrow 0$
  4. $V^{t+1} \leftarrow (1-\alpha)SD^{-1}V^t + \alpha R$ and $t \leftarrow t+1$
  5. Go to step 4 unless $V^t$ has converged
  6. $Y_i^U \leftarrow argmax_c(V_{ci})$
```

HF finds the probability of *hitting* (landing for the first time on) a labeled node in class c if we start walking randomly from an unlabeled node; with respect to labeled nodes, this is propagation via *reverse random walks*, or *diffusing*. MRW finds the probability of landing on a node by walking from labeled nodes in c, with a fixed probability of "teleporting" back to c at each step; this is propagation via *random walk with restart*. Classification experiments comparing these two methods on network datasets [13] have shown that, while the classification accuracies are similar with a large number of *seeds* (labeled instances), MRW consistently outperforms HF when seeds are scarce and MRW seems more consistent with seeds of varying quality.

2.2 Implicit Manifolds

Differences aside, an important algorithmic similarity between HF and MRW (as shown in Algorithms 1 and 2) is that both have as their core operation iterative matrix-vector multiplications (step 2 and 4, respectively). This allows us to replace the costly dense matrix as in Equation 1, and the iterative operations for HF and MRW become:

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

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

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

Now instead of a dense matrix-vector multiplication with n^2 space and time requirements, we have a series of sparse matrix-vector multiplication linear to n. Before plugging these into the algorithms, we need to do one more thing: we need to find the diagonal matrix D^{-1} without S. It follows 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 decomposition: compute a vector $\mathbf{d} = FF^T\mathbf{1}$, where $\mathbf{1}$ is a vector of 1's, and let $D(i,i) = \mathbf{d}(i)$. Thus the implicit manifold S is constructed through mathematical equivalence, resulting in a simple, fast, and space-efficient algorithms that yield the exact same solutions.

Cosine Similarity Manifold

If we view the rows of F as feature vectors in vector space, then path-folding is equivalent to the *inner product similarity* of instances. However, this is just one of many functions used for measuring data point similarity. For example, if the instances represent documents and the features are words, one may want to normalize the feature vectors by its length.

It turns out that the implicit manifold can be easily constructed with other similarity functions, *as long as the manifold can be represented with a series of sparse matrix multiplications*. Here we consider *cosine similarity* [12, 17], widely used for comparing document similarity: $cos(\mathbf{a},\mathbf{b}) = \frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|}$ where cos(**a**,**b**) is simply the cosine of the angle between vectors $\mathbf{a}$ and $\mathbf{b}$. For the normalizing term $1/(\|\mathbf{a}\|\|\mathbf{b}\|)$, we need to calculate an additional diagonal matrix $N_{cos}(i,i) = 1/\sqrt{(F(i)F(i)^T)}$ where $F(i)$ is the ith row-vector of F. Following inner product similarity the values of the diagonal matrix D can be computed by $\mathbf{d} = N_cFF^TN_c\mathbf{1}$. Then the iterative operations for HF and MRW become:

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

As before, all operations in constructing N_c and D are sparse matrix-vector multiplications. Note that while we *could* pre-process 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 try; calculating similarity functions on-the-fly will often prove to be a much more efficient approach.

Bipartite Graph Walk Manifold

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. Here S is defined as $FD_c^{-1}F^T$, where D_c is the diagonal degree matrix for the columns of F (as opposed to D, the diagonal degree matrix for the rows of F), resulting in the following modified iterative operations for HF and MRW:

$$\begin{aligned} \text{HF:} \quad & V^{t+1} \leftarrow D^{-1}(F(D_c^{-1}(F^TV^t))) \qquad (2) \\ \text{MRW:} \quad & V^{t+1} \leftarrow (1-\alpha)F(D_c^{-1}(F^T(D^{-1}V^t))) + \alpha R \end{aligned}$$

This manifold can be interpreted in two ways. It simulates a random walk on F as a bipartite graph, where each iteration is equivalent to taking two Markovian steps—the first step walks from the instances to the features, and the second step walks from the features back to the instances. This similarity function 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 literature [17]. They are equivalent if we replacing the inverse document frequency (IDF) with inverse collection frequency—the total weighted count of a feature in the entire dataset.

3. EXPERIMENTS

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 1, 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 [17]. For NP-context data, this particular task of NP categorization has been performed successfully using co-EM [19], which is closely related to the bipartite graph walk manifold. In fact, the harmonic functions method with bipartite graph walk manifold is exactly equivalent [1] to the co-EM algorithm used for performing information extraction from free text proposed by [11].

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 1: 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. * indicates that the memory requirement is estimated using random sampling and extrapolation.**

3.1 Document Categorization

The 20 Newsgroups dataset (20NG) is a collection of approximately 19K newsgroups documents, roughly 1,000 per newsgroup [18]. The class labels are the newsgroups groups; the features are word tokens, weighted according to the log-normalized TF-IDF scheme [17, 12]. The Reuters Corpus Volume 1 (RCV1) dataset is a benchmark collection of 804K newswire stories [12]. 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 [6]. 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 [17]. 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 1.

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.[2] 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.

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 [13]. 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.

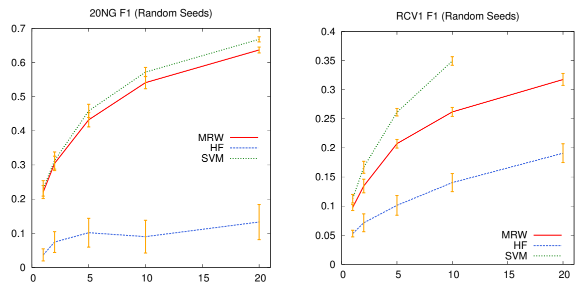


Figure 1: F1 scores on the 20NG and RCV1 datasets. The x-axis indicates the number of labeled instances and the y-axis indicates the macro-averaged F1 score. Vertical lines indicate standard deviation (over 20 trials for 20NG and 10 for RCV1) using randomly selected seed labels.

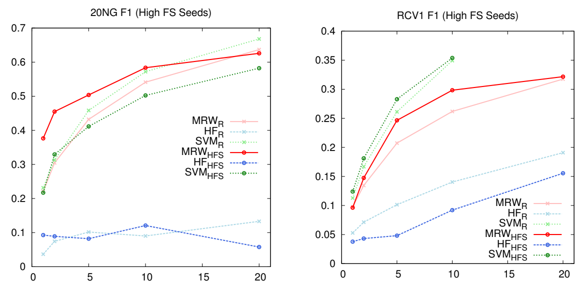


Figure 2: F1 scores on the 20NG and RCV1 datasets using preferred (high feature weight sum) seeds. Subscript HFS indicates result using high feature-sum seeds and R indicates result using random seeds—included for comparison.

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 2. While HF does not seem to improve at all, MRW does improve on both datasets, with more dramatic improve

[1]Refer to Appendix A for details.

[2]HF results can be improved by re-weighted output probability; but this introduces a not-well-understood parameter and is beyond the scope of this paper.

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.

3.2 Noun Phrase Categorization

The City and 44Cat datasets are derived from the NP-context data described in [4]. 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 A), which worked well on this type of data [11]. 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 [2]; 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 [5], we also want to see 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 β to MRW so that step 4 in Algorithm 2 becomes:

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

where $\alpha+\beta \leq 1$. Typically β is very small, and can be considered an uniform smoothing factor on the confidence. Note that the label prediction does not depend on β 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 2: 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 2 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.

Method	SVM	HF	MRW	HF	MRW
Manifold	-	inner	inner	bipart	bipart
NDCG	0.0263	0.0402	0.0405	0.0406	**0.0408**
AP	0.0208	0.6728	0.7067	0.7130	**0.7389**
P@10%	0.0123	0.8732	0.8926	0.8796	**0.9094**
P@20%	0.0143	0.8698	0.8991	0.8941	**0.9162**
P@30%	0.0168	0.8773	0.9093	0.9036	**0.9116**
P@40%	0.0199	0.8574	0.8957	0.9118	**0.9179**
P@50%	0.0210	0.8227	0.8647	0.8832	**0.9038**
P@60%	0.0236	0.7591	0.7990	0.8093	**0.8307**
P@70%	0.0265	0.6337	0.6743	0.6805	**0.7189**
P@80%	0.0267	0.4131	0.4533	0.5087	**0.5297**
P@90%	0.0272	0.1927	0.2155	0.2521	**0.2926**
P@100%	0.0274	0.0275	0.0279	0.0280	**0.0289**

Table 2: City dataset result. Boldfaced font indicates the highest number in a row. *inner* refers to the inner product manifold and *bipart* refers to the bipartite graph walk manifold. Note that HF with bipart is equivalent to co-EM as used in [11]

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 [5]. 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 3.2. Here we see that again SVM does poorly compared to graph-based SSL 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). A breakdown of the result by category can be found in Appendix B.

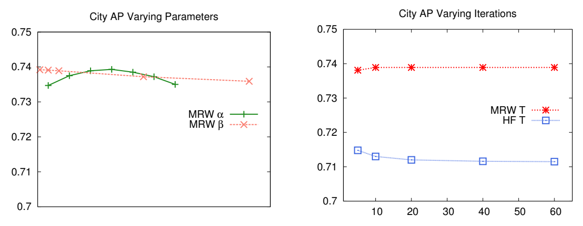
Implicit manifolds also yield fast runtimes. The smallest dataset (20NG) takes less than 1 second for both HF and MRW, and the largest (44Cat) takes less than 3 minutes. We did not try running explicit manifold versions of the algorithm for runtime comparison because that would require more memory than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

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 3: 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. Note that no statistical significance is found between HF and MRW.

3.3 Parameter Sensitivity

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 α, β, and the number of iterations T (correlated with the convergence threshold). The only parameter for HF is T. We plot varying parameters and their effect on average precision in Figure 3. Here neither method appears to be very sensitive. [3]



(a) α and β (b) the # of iterations T

Figure 3: Parameter sensitivity. The x-axis correspond to parameter values and the y-axis shows average precisions. α ranges from 0.05 to 0.65, β ranges from 0.0001 to 0.01; the number of iterations T are indicated below x-axes.

4. CONCLUSIONS

We have introduced a set of tools for efficient semi-supervised learning on text data with implicit manifolds. We considered two label propagation methods (HF and MRW) and three similarity functions (inner product, cosine similarity, and bipartite graph walk), leading to five new learning methods and a new interpretation of one old method (as HF with bipartite walks is identical some definition of co-EM). Experiments on document categorization with cosine similarity seem to favor MRW over HF, and show that MRW is competitive with, and sometimes superior to, supervised SVM. On NP categorization, all the SSL methods far outperform SVM, and are comparable with each other one large dataset, while on a smaller, more completely-labeled dataset, MRW bipartite-walk performs best.

Beyond these experimental data points, the more important contribution of this paper is a general framework in which graph-based semi-supervised learning methods can be very efficiently applied to general data, without pre-computing pairwise distances between instances. Avoiding explicit computation of pairwise distances means that the methods considered here are quite different in computational complexity from label-propagation methods applied to (for instance), a k-NN graph over all instances; in particular, all space and time requirements in our framework are *strictly linear* in the size of the input data. The availability of simple, scalable implicit manifold construction means that many SSL methods can now be efficiently applied to a wide variety of large text datasets, and provides an alternative to techniques that sparsify similarity matrices [27] in order to speed up SSL methods. [4]

5. ACKNOWLEDGMENTS

This work is supported by NSF grant IIS-0811562, NIH grant R01GM081293, and a gift from Google.

6. REFERENCES

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

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

[3] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. 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.

[4] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the 24th Conference on Artificial Intelligence*, 2010.

[5] A. Carlson, J. Betteridge, R. C. Wang, E. R. H. Jr., and T. M. Mitchell. Coupled semi-supervised learning for information extraction. In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining*, 2010.

[6] W.-Y. Chen, Y. Song, H. Bai, C.-J. Lin, and E. Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[7] B. Gallagher, H. Tong, T. Eliassi-Rad, and C. Faloutsos. Using ghost edges for classification in sparsely labeled networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

[8] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen. Web content categorization using link information. Technical report, Stanford University, 2006.

[9] T. Haveliwala, S. Kamvar, and G. Jeh. An analytical comparison of approaches to personalizing pagerank. Technical report, Stanford University, 2003.

[10] J. He, J. Carbonell, and Y. Liu. Graph-based semi-supervised learning as a generative model. In *International Joint Conferences on Artificial Intelligence*, 2007.

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

[12] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[13] F. Lin and W. W. Cohen. Semi-supervised classification of network data using very few labels. In *Proceedings of the 2010 International Conference on Advances in Social Networks Analysis and Mining*, 2010.

[3]Note that HF's AP peaks around $T = 5$ and it actually *degrades* with more iteration. This suggests early-stopping HF may yield better performance—hence why we fixed $T = 10$ for HF. This also points to an advantage MRW has over HF—unlike HF, MRW does not "over-iterate".

[4]We note that for some datasets sparsified manifold constructions such as a k-nearest neighbor graph may help to reduce noise and therefore have merit in increase classification accuracy; however, it comes at a price of additional construction time.

[14] F. Lin and W. W. Cohen. A very fast method for clustering big text datasets. In *Proceedings of the 19th European Conference on Artificial Intelligence*, 2010.

[15] W. Liu, J. He, and S.-F. Chang. Large graph construction for scalable semi-supervised learning. In *Proceedings of the International Conference on Machine Learning*, 2009.

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

[17] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[18] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[19] K. Nigam and R. Ghani. Analyzing the effectiveness and applicability of co-training. In *Proceedings of the 9th International Conference on Information and Knowledge Management*, 2000.

[20] P. Sarkar and A. W. Moore. Fast dynamic reranking in large graphs. In *Proceedings of the 18th International World Wide Web Conference*, 2009.

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

[22] P. P. Talukdar, J. Reisinger, M. Paşca, D. Ravichandran, R. Bhagat, and F. Pereira. Weakly-supervised acquisition of labeled class instances using graph random walks. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, 2008.

[23] H. Tong, C. Faloutsos, and J.-Y. Pan. Fast random walk with restart and its applications. In *Proceedings of the 2006 IEEE International Conference on Data Mining*, 2006.

[24] K. Zhang, J. T. Kwok, and B. Parvin. Prototype vector machine for large scale semi-supervised learning. In *Proceedings of the International Conference on Machine Learning*, 2009.

[25] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Scholkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems 16*, 2004.

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

[27] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *The 20th International Conference on Machine Learning*, 2003.

APPENDIX

A. RELATION AND EQUIVALENCE TO 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 [11].

A co-EM algorithm [11] is shown Figure 3, where $\hat{f}_n$ and $\hat{f}_c$ are the two classifiers corresponding to two views of the data (n and c). If we let $\mathbf{v}^t$ be a vector indicating $\hat{f}_n$ and F be the matrix form of co-occurrence data N, then step 2 in Figure 3 can be computed by $D_c^{-1}F\mathbf{v}^t$, and step 3 can in turn be computed by $D^{-1}F(D_c^{-1}F\mathbf{v}^t)$, which is equivalent to a single-class version of Equation 2. 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 3 The co-EM algorithm from [11]

```
**Input:** Data $N$ where $N(i,j)$ indicates the co-occurrence count of $(n_i, c_j)$; positive labels $L$
**Output:** probability $\hat{f}_n$ of an instance being positive

  1. initialize $\hat{f}_{n0}(n_i) = \begin{cases} 1 & \text{if } n_i \in L \\ 0 & \text{otherwise} \end{cases}$
  2. $\hat{f}_c(c_j) = \frac{\sum_{n_i} \hat{f}_n(n_i) * N(i,j)}{\sum_i N(i,j)}$
  3. $\hat{f}_n(n_i) = \begin{cases} 1 & \text{if } n_i \in L \\ \frac{\sum_{c_j} \hat{f}_c(c_j) * N(i,j)}{\sum_j N(i,j)} & \text{otherwise} \end{cases}$
  4. Go to step 2 unless $\hat{f}_n$ has converged
```

B. 44CAT DATASET RESULT BY CATEGORY

We break down the results of the estimated precision at top $1,000$ retrieved NPs into 44 categories in Figure 4. 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 $1,000$.

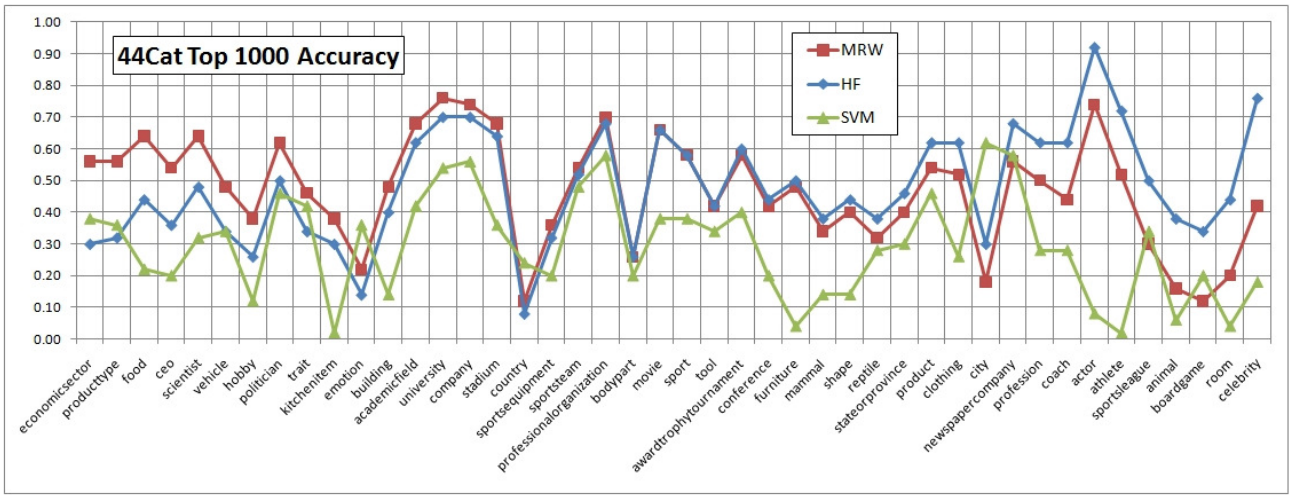


Figure 4: Sampled per-category accuracies of the top 1000 retrieved NPs on the 44Cat dataset. The categories are ordered from left to right according to the difference between the MRW accuracy and HF accuracy, from the high to low.