Machine Learning for Efficient Neighbor Selection in Unstructured P2P Networks

Robert Beverly
MIT CSAIL
rbeverly@csail.mit.edu

Mike Afergan
Akamai/MIT
afergan@alum.mit.edu

ABSTRACT
Self-reorganization offers the promise of improved performance, scalability and resilience in Peer-to-Peer (P2P) overlays. In practice however, the benefit of reorganization is often lower than the cost incurred in probing and adapting the overlay. We employ machine learning feature selection in a novel manner: to reduce *communication cost* thereby providing the basis of an efficient neighbor selection scheme for P2P overlays. In addition, our method enables nodes to locate and attach to peers that are likely to answer *future queries* with no a priori knowledge of the queries.

We evaluate our neighbor classifier against live data from the Gnutella unstructured P2P network. We find Support Vector Machines with forward fitting predict suitable neighbors for future queries with over 90% accuracy while requiring minimal (<2% of the features) knowledge of the peer's files or type. By providing a means to effectively and efficiently select neighbors in a self-reorganizing overlay, this work serves as a step forward in bringing such architectures to real-world fruition.

1. INTRODUCTION
Simple unstructured Peer-to-Peer (P2P) networks are both popular and widely deployed [8, 10]. Nodes issue queries that propagate through the overlay and receive answers from peers able to satisfy the query. Because unstructured networks allow nodes to interconnect organically with minimal constraints, they are well-suited to self-reorganization. For example, prior research investigates reorganization for improved query recall, efficiency and speed, as well as increased system scalability and resilience [1, 2, 3, 11, 13].

In practice however, the benefit of reorganization is often lower than the cost in a classic exploration versus exploitation paradox. A critical question prior research does not address is how nodes within a self-reorganizing P2P system can determine the suitability of another node, and hence whether or not to connect, in real-time. Nodes must classify potential peers as good or poor attachment points both effectively (with high success) and efficiently (with few queries).

Given an omniscient oracle able to determine a node's future queries and the fraction of those queries matched by other nodes, neighbor selection is readily realizable. Naturally, nodes do not have access to such an oracle. This work seeks to understand how to emulate, in a distributed fashion with minimum communication cost, the functionality of an online oracle as a component of a self-reorganizing network. In particular, nodes within a self-reorganizing network face two primary challenges: minimizing load induced on other network participants and locating neighbors that are likely to answer *future* queries.

We abstract these challenges into a distributed, uncoordinated, per-node machine learning classification task. Based on minimal queries, nodes predict whether or not to connect to a peer. A key insight of this work is that minimizing the induced load on other nodes maps to a feature selection problem. We wish to build an effective classification model by finding a small set of highly discriminatory queries.

Our analysis uses live Gnutella data to understand the efficacy and parameterization of machine learning techniques for neighbor selection. We determine the accuracy, precision and recall performance of Support Vector Machines (SVMs) [14] and empirically find a training size with high prediction performance. We then experiment with forward fitting and mutual information feature selection algorithms as a means of finding queries with high discrimination power. The major contributions of our research are:

1. Novel application of machine learning to the neighbor selection problem in self-reorganizing networks.

2. Correct neighbor prediction with over 90% accuracy while requiring minimal queries (<2% of features). Our technique efficiently chooses neighbors that successfully answer not only current, but also future queries.

Consequently, we hope this paper serves as a step forward for research in the space of self-reorganizing overlays by providing a scalable means of neighbor selection.

2. RELATED WORK
Unstructured overlays can adapt, grow organically and reorganize dynamically. Earlier work [9] recognized that, if locality exists, preferentially connecting peers with similar interests together in unstructured P2P networks can minimize query flooding. The work of Sripanidkulchai, et. al [11] relies on the presence of interest-based locality to create "shortcuts." Each peer builds a shortcut list of nodes that answered previous queries. To find content, a peer first queries the nodes on its shortcut list and, only if unsuccessful, floods the query. These systems present promising reorganization methods within unstructured P2P networks, but do not address the neighbor selection task.

In the domain of structured overlays, Tang, et. al propose pSearch [13] as a means to semantically organize nodes. Within their semantically organized network, search proceeds using traditional information retrieval methods that return results based on document similarity, not just keywords. Several pieces of work [4, 6] examine neighbor selection within structured overlays, particularly the impact of routing geometry and the ability to choose neighbors in different structured overlays for resilience and proximity.

In many respects, our problem is most akin to text categorization and we draw upon the rich literature of prior work espousing machine learning theory. One of the earlier works in application of SVMs to text categorization is from Joachims [7]. Yang and Liu examine several text categorization methods against standard new corpora, including SVMs and Naïve Bayes [15]. Our results similarly find SVMs outperforming Naïve Bayes for our application; we omit Naïve Bayes results in this paper. Finally, as in Yang's comparative study [16], forward fitting also outperforms mutual information feature selection on our dataset.

3. NEIGHBOR SELECTION

Finding peer nodes in unstructured P2P networks is accomplished in a variety of ways. For instance in Gnutella-like overlays, bootstrap nodes maintain pointers into the network while every node advertises the addresses of its neighbors. However, it is not obvious how a node can, in real-time, determine whether or not to connect to another node. Exploring other nodes incurs cost and presents a paradox: *the only way to learn about another node is to issue queries, but issuing queries makes a node less desirable and the system less scalable.*

A key insight of our research is that efficient neighbor selection maps to machine learning feature selection. We ask: "for a node i, does there exist an efficient method, i.e. a small set of key features, by which i can optimize its choice of neighbors?" While feature selection is traditionally used in machine learning problems to reduce the computational complexity of performing classification, we use it in a novel way. Specifically, we use feature selection to *minimize the number of queries*, which equates to network traffic and induced query load.

In this section, we concretely describe the neighbor selection problem in the context of self-reorganizing P2P systems. We start by giving details of our dataset. We then formulate the problem and data as a machine learning task.

3.1 Live P2P Datasets

We experiment on real, live Gnutella datasets from two independent sources: our own measurement of 1,500 nodes[1] and a public, published repository of approximately 4,500 nodes from Goh, et al. [5]. Both datasets include timestamped queries and files offered across all nodes. The collection methodology is similar for both datasets. A capture program establishes itself as a Gnutella UltraPeer and promiscuously gathers queries and file lists from all leaf node connections.

[1]Publicly available from:
`http://ana.csail.mit.edu/rbeverly/gnutella/`

While our datasets focus on Gnutella, we believe that they are sufficiently representative of general P2P usage (in particular, the Gnutella network is estimated to contain approximately 3.5M users [10]). Detailed inspection of the datasets reveals a wide range of searches and content including music, videos, programs, images, etc of all sizes and type. While other popular P2P file sharing networks exist, it is reasonable to believe that most general-purpose networks will see comparable usage patterns. Many additional motivations for using Gnutella as a reference P2P network, including size, scope and growth, are given in [12].

We tokenize queries and file names in data by eliminating: i) non-alphanumerics; ii) stop-words: "it, she, of, mpeg, mp3," etc.; and iii) single character tokens. The resulting tokens are produced by separating on white-space. We assume tokenization in the remainder of this work. Let N be the set of all nodes and $n = |N|$. Let $\mathbf{q}_i$ and $\mathbf{f}_i$ represent the tokenized queries and file store of node i respectively. We represent the set of all unique tokens across the queries and files as $Q = \bigcup \mathbf{q}_i$ and $F = \bigcup \mathbf{f}_i$.

Encouragingly, our experiments yield similar results using either dataset, lending additional credence to our methodology. Due to space constraints however, the results presented in this paper are limited to the Goh dataset.

3.2 Representing the Dataset

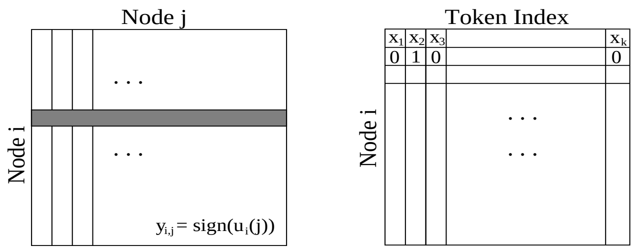
To qualitatively compare peers, we introduce the notion of a utility function. Given $\mathbf{q}_i$ and $\mathbf{f}_i$ for every node i, we can evaluate whether a potential neighbor has positive utility, i.e. nodes are individual, selfish utility maximizers. Utility may be a function of many variables including induced query load, query success rate, etc. However, we are primarily interested not in the specific mechanics of a utility-based self-reorganizing network, but rather the neighbor selection task. Therefore, in this work, we define $u_i(j)$, node i's utility in connecting to node j, simply as the number of successful queries from i matched by j. A single query from i matches a single file held by j if and only if all of the query tokens are present in that file[2].

We represent the dataset with the two matrices in Figure 1, an adjacency and word token matrix:

- *Adjacency Matrix* $\mathbf{Y}$: An n-x-n pair-wise connectivity matrix where $\mathbf{Y}_{i,j} = \text{sign}(u_i(j))$. Because our dataset includes all queries and files of every node, our omniscient simulator definitively knows how many queries of each node are matched by every other peer. Thus, $\mathbf{Y}_{i,j} = +1$ indicates that node i wants to connect to node j.

- *File Store Matrix* $\mathbf{X}$: Using all file store tokens, F, we assign each token a unique index k. The word token boolean matrix indicates the presence or absence of a given file token for every node in the system. $\mathbf{X}_{i,j} = 1 \iff F_j \in \mathbf{f}_i$.

[2]We also used more sophisticated decreasing marginal utility functions that consider both query matches and induced system load along with an ϵ-equilibrium analysis for non-strict utility maximizers, but omit results here.



(a) Adjacency Matrix (b) File Store Matrix

Figure 1: (a) The binary n-x-n adjacency matrix indicates whether node i wishes to connect to node j based on utility $u_i(j)$. (b) We assign a unique index k to all file store tokens and form a boolean per-node word token presence matrix X.

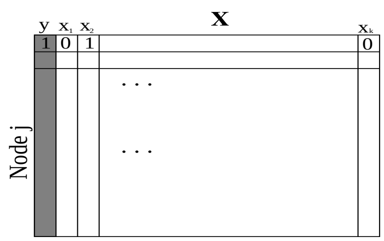


Figure 2: Representing the *single node* i: The i'th row of the adjacency matrix (fig 1a) is the first column (shaded) and represents node i's connection preferences (class labels). To this the file store token matrix (X) is horizontally concatenated.

From the adjacency and file store matrices we create a per-node matrix, $[\mathbf{Y}(i,:)^T, \mathbf{X}]$, as shown in Figure 2. Note that we compute the adjacency and file store token matrices explicitly only in order to evaluate the performance of our neighbor selection algorithm; our scheme **does not** require complete knowledge of all files and queries in the network. Rather, we employ the omniscient oracle only to evaluate the performance of our neighbor prediction algorithm.

3.3 Formulating the Learning Task

Given the hypothetical off-line (oracle) representation of a node's state as depicted in Figure 2, we now turn to the problem of classification. Note that the problem we face is slightly non-standard – we have a separate classification problem for each node. That is, the features that are optimal can, and likely will, be different from node to node. In addition, the optimal features need not match the node's queries. For instance, while a node may issue queries for "lord of the rings," the single best selective feature might be "elves." This example provides some basic intuition of how our system finds peers that are capable of answering future queries. By connecting to peers using "elves" as a selection criterion, a future query for "the two towers" is likely to succeed given interest-based locality.

Figure 3 shows how the conjoined matrices as shown in Figure 2 are split and used as input to machine learning algorithms. Some number of nodes, significantly fewer than the total number of nodes n, are selected at random to serve as training samples. The learner is given the word token features present for each training node ($\hat{\mathbf{X}}$ a row subset of $\mathbf{X}$) along with the corresponding classification labels ($\mathbf{y}$). For our neighbor selection task, the label is a binary decision variable, $y \in \{\pm 1\}$ where $y = +1$ indicates a good connection and $y = -1$ a poor connection. We consider the size and complexion of the training data in the next Section. Using the training data, the learner develops a model that uses a small number of features $\boldsymbol{\theta} \in \hat{\mathbf{X}}$ in order to predict future connection decisions. We evaluate the efficacy of this model against the test data, i.e. whether the model correctly predicts the unknown y connection labels in the test set. Thus, in the testing phase, the input to the classifier is the small number features $(\theta_1, \cdots, \theta_d)$ where $d \ll k$, without either the labels (y) or the full word tokens $(x_1, \cdots, x_k)$.

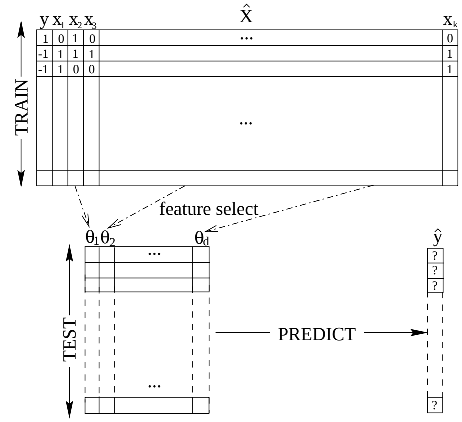


Figure 3: The machine learning task: from randomly selected training samples, find the best model and features ($\boldsymbol{\theta} \subset \hat{\mathbf{X}}$) to minimize training error. With this small set ($d \ll k$) of features, predict neighbor suitability (y class label).

Notice that the features we train and test on do not include any queries Q from our dataset. Only the y labels depend on the queries. Thus, successful connection predictions imply the ability to predict queries *yet to be asked.*[3]

We find that some nodes in the dataset have queries that are matched by very few other nodes. Therefore, a prediction model that deterministically predicts not to connect will yield a high accuracy – and thus a misleading result. Consider a node whose query is matched by only 1 of 500 other nodes. A classifier that always predicts not to connect gives an accuracy of 499/500 = 99.8%. To better assess our neighbor selection scheme without bias or skewing the results, we randomly select 50 nodes that have at least 20% positive labels, i.e. a non-trivial number of suitable potential peers. In this way, we choose to evaluate the nodes that are **most difficult** to accurately perform predictions with and thereby stress our approach.

[3]Additionally, our prediction accuracy implies correlation between a single node's file store and queries, a result we analyze in detail in [2].

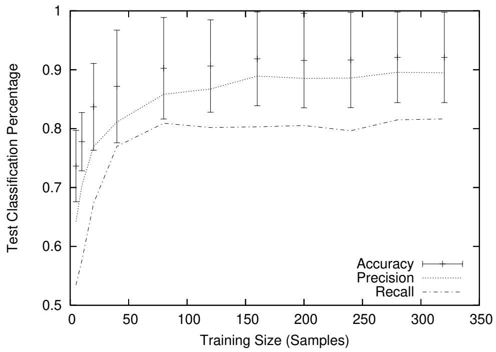


Figure 4: SVM Neighbor prediction: classification performance versus number of training samples

These nodes include $k = 37,000$ unique file store tokens. We provide both average and variance measures across these nodes from our dataset. Thus, we wish to show that our scheme is viable for the vast majority of all nodes.

We note that incorrect predictions in our scheme are not fatal. In the overlays we consider, a node that attaches to a peer who in fact provides no utility can simply disconnect that peer. Therefore, while the statistical methods we employ provide only probabilistic measures of accuracy, they are well-suited to the neighbor selection task.

3.4 Methodology Summary

Our evaluation methodology simulates the following algorithm on nodes from our dataset and measures prediction accuracy. For a node $i \in N$ in the system that wishes to optimize its connection topology:

1. Randomly select $T \subset N$ other peers as trainers, where $|T| \ll |N|$
2. Receive file tokens $\mathbf{x}_t$ from each $t \in T$
3. Determine utility of each training peer $y_t = \text{sign}\,(u_i(t))$
4. Let $X = \bigcup \mathbf{x}_t$ and $k = |X|$
5. Find features $\theta_1, \cdots, \theta_d \subset X$, where $d \ll k$, which best predict $\hat{y}_t = y_t \forall t \in T$
6. Issue $\boldsymbol{\theta}$ to test set $M \in N - T$, predict whether to connect to each peer $j \in M$

4. SUPPORT VECTOR MACHINES

We obtain the best neighbor selection prediction results using SVMs. SVM classifiers [14] find an optimal separating hyperplane that maximizes the margin between two classes of data points. The hyperplane separator is orthogonal to the shortest line between the convex hulls of the two classes. Because this separator is defined only on the basis of the closest points on the convex hull, SVMs generalize well to unseen data. Additional data points do not affect the final solution unless they redefine the margin. While the separator may be linear in some higher-dimensional feature space, it need not be linear in the input space. Determining the separating hyperplane for the data in some high dimensional space is performed via constrained optimization.

SVMs are a natural selection for neighbor selection since the problem is separable, we may need to consider high-dimensional data, and the underlying distribution of the data points and independence are not fully understood. We use the MySVM package for our experiments [7].

To reduce possible dependence on the choice of training set, all results we present are the average of five independent experiments. We randomly permute the order of the dataset so that, after splitting, the training and test samples are different between experiments. In this way, we ensure generality, a critical measure of learning effectiveness. We evaluate model performance on the basis of classification accuracy, precision and recall.

Accuracy is simply the ratio of correctly classified test samples to total samples. *Precision* measures the number of positive predictions that were truly positive. Finally, *recall* gives a metric of how many positive samples were predicted positive. All three measures are important in assessing performance; for instance, a model may have high accuracy, but poor recall as mentioned in the previous section when nodes have few suitable peers in the entire system.

4.1 Number of Training Points

An important first consideration is the sensitivity of the model to the number of training points (step 1 in §3.4). In Figure 4 we plot the test accuracy, precision and recall versus the number of training points.

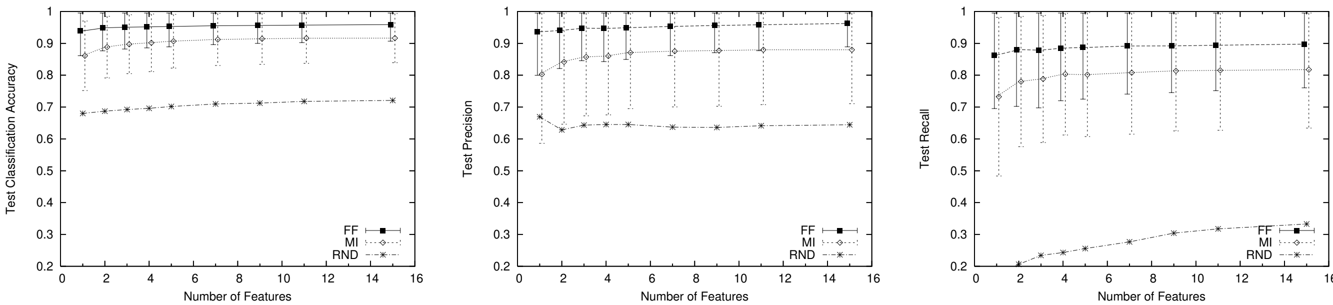
Test error decreases significantly in the range [10,70]. Test error is minimized around 150 points, after which a small amount of over-fitting is present. Beyond 100 training points, accuracy, precision and recall all remain fairly stable. Because we are interested in obtaining the best combination of the three performance metrics while minimizing the number of training points, we select 100 training points for the remainder of our experiments.

4.2 Feature Selection

In order to find a small number of highly discriminative features (step 5 in §3.4), we turn to feature selection methods [16]. Recall that we use feature selection in a novel manner, not to reduce the computation complexity of classification, but rather to minimize communication cost for neighbor selection in the network. The first feature selection method we consider is mutual information (MI). Mutual information attempts to use combinations of feature probabilities to assess how much information each feature, i.e. word token, contains about the classification. MI evaluates the mutual information score $I(\theta_i; y)$ for each feature $\theta_i \in \hat{\mathbf{X}}$ and sequentially picks the highest scoring features independently of the classifier. The MI score is in the range $[0, 1]$ and will be zero if the feature is completely independent of the label or one if they are deterministically related.

$$I(\theta_i; y) = \sum_{\theta_i \in \{0,1\}} \sum_{y \in \{\pm 1\}} \hat{P}(\theta_i, y) \log_2 \frac{\hat{P}(\theta_i, y)}{\hat{P}(y)\hat{P}(\theta_i)} \qquad (1)$$

Secondly, we use greedy forward fitting (FF) feature selection. Forward fitting feature selection simply finds, in succession, the next single feature that minimizes training error. Therefore, training error decreases monotonically with the



(a) Neighbor Prediction Accuracy (b) Neighbor Prediction Precision (c) Neighbor Prediction Recall

Figure 5: Neighbor selection prediction performance for three different feature selection algorithms: Mutual Information (MI), Forward Fitting (FF), and Random (RND) with an SVM model. The points represent average performance across nodes in our dataset, while the error bars show the standard deviation.

number of features. For an error function $V(f(\boldsymbol{\theta}), \cdot)$, find the next feature θ_i from the remaining features not previously selected in $X_1, \ldots, X_{i-1}$. Thus, we evaluate each potential feature i for the fixed set of $i-1$ features. We use SVM accuracy as the error function $f(\cdot)$, although forward fitting can be used with any model and error function. Formally:

$$\theta_i \leftarrow \operatorname*{argmin}_j V(f(\hat{\mathbf{X}}, x_j), \mathbf{y}) \forall x_j \notin X_1, \ldots, X_{i-1} \quad (2)$$

Forward fitting is computationally expensive because it requires computing a combinatorial number of possible feature combinations. Mutual information is much faster, but does not always find the optimal features. A weakness of MI is that it may choose two features that are themselves closely dependent and thus the second feature provides little additional classification power. In contrast, forward fitting will continually seek the next best performing feature without this potential dependence.

4.3 Prediction Results

Figure 5 summarizes the most meaningful results from our feature selection and SVM experiments (step 6 in §3.4). The experiment is run five times for each configuration and the average numbers are presented in the graph along with the standard deviation error range. We include random feature selection as a baseline measure. The primary observations from these graphs are:

- *We are able to make accurate predictions with as few as 5 features.* This is a surprisingly good result. Recall that we are handicapped from the fact that a) we consider only the file features not the query features, even though the queries are what generated the results; and b) there are 37,000 single-word features and thus a massive number of multi-word combinations.
- *Forward-fitting performs better than mutual information.* In particular, we see that even with one feature, FF has a lower test error than MI.
- *The random feature selector performs poorly.* While we expect randomly chosen features to perform the worst, it provides validation of our results and methodology and proved to be a surprisingly useful benchmark in practice when building our system.

5. DISCUSSION

Here we examine the larger conclusions that can be drawn from our findings:

While our input space is of high dimension, it can be effectively summarized with very few parameters. The summarization aspect of our problem inspired us to consider an SVM approach. Forward fitting reveals that a small number of features effectively correlates to accurate predictions. Our findings are in stark contrast to the standard motivation for SVMs in text classification problems where the inputs are independent and the primary motivation for SVMs is to reduce the computational complexity.

SVMs allow us to accurately model the problem with little or no underlying understanding of the inputs. While we have a general understanding of the data, we face the challenge that we do not totally understand the relationship between the features and moreover the nature of these relationships vary from node to node. Nodes in a real system face this same challenge. Therefore SVMs, which make no underlying assumptions regarding the data, are particularly well-suited for our problem.

For our problem, forward fitting outperforms mutual information. The literature is mixed on the relative merit of mutual information and forward fitting. In our problem, we were interested in seeing if, as enough features were added via MI, the combination of features could outperform FF, where features are selected in a greedy fashion. Empirically this was not the case.

One reason FF performs well is the high degree of correlation between features in the dataset. For example, if a user has songs by "Britney Spears" both "Britney" and "Spears" may be descriptive features. However, simple MI will not take into account that once it adds "Britney", adding "Spears" will not improve prediction performance. In future work, we plan to investigate the ability to remove correlated features found via MI by computing feature-to-feature MI. Conversely, forward fitting will likely only add one of these terms, moving on to a more descriptive second term. The danger of forward fitting of course is over-fitting but we do not observe over-fitting in practice.

For our problem, SVMs do not suffer significantly from over-fitting. As witnessed by varying the number of training points, SVMs are robust against over-fitting. While some over-fitting is present in our empirical results with forward fitting, in the context of our particular problem it has little impact. In particular, we are attempting to minimize the number of features with as little impact on prediction performance as possible. Therefore, the fact that too many features leads to worse classification performance is not as problematic as it may be for other problems.

Our neighbor selection algorithm is computationally practical. Nodes can use the SVM prediction we describe in a completely decentralized fashion. While forward fitting gives the highest accuracy, it requires training many SVMs. Nodes with limited computational power can use MI to achieve comparable accuracy. In future work, we may consider FF over multiple features at once. While this method of forward fitting may be more expensive computationally, it could run as a background process on a user's desktop (say overnight) and thus not be prohibitively expensive in practice.

Our neighbor selection algorithm is practical in real networks. While we simulate and model the operation of nodes using machine learning algorithms to predict suitable neighbors, our scheme is viable in practice. P2P systems, and Gnutella in particular, utilize a system of caches which store IP addresses of nodes in the overlay thereby allowing new clients to locate potential peers. Our experiments show that randomly selecting $\simeq 100$ peers on which to train suffices to build an effective classifier. Because we randomly permute the set of training nodes in each experiment, the density of "good neighbors" in 100 peers is sufficiently high for accurate future predictions.

Efficient neighbor selection is a general problem. While we focus only on P2P overlays, minimizing the communication overhead via feature selection methods such as those in our algorithm may generalize to tasks in other networks or applications.

6. CONCLUSIONS

In this work, we examined efficient neighbor selection in self-reorganizing P2P networks. While self-reorganizing overlays offer the potential for improved performance, scalability and resilience, their practicality has thus far been limited by a node's ability to efficiently determine neighbor suitability. We address this problem by formulating neighbor selection into a machine learning classification problem. Using a large dataset collected from the live Gnutella network, we examined the efficacy of Support Vector Machine classifiers to predict good peers. A key insight of our work was that nodes can use feature selection methods in conjunction with these classifiers into order to reduce the communication cost inherent in neighbor selection. Using forward fitting feature selection, we successfully predicted suitable neighbor nodes with over 90% accuracy using only 2% of features.

By addressing the efficiency of neighbor selection in the formation of self-reorganizing networks, we hope our work serves as a step forward in bringing self-reorganizing architectures to real-world fruition.

Acknowledgments

We thank Steve Bauer, Bis Bose, Dave Clark, Peyman Feratin, Simson Garfinkel, Karen Sollins and our reviewers for invaluable feedback. Rob Beverly was supported in part by Cisco Systems and NSF Award CCF-0122419.

7. REFERENCES

[1] A. Asvanund, S. Bagla, M. Kapadia, R. Krishnan, M. D. Smith, and R. Telang. Intelligent club management in peer-to-peer networks. In *Proceedings of First Workshop on Economics of P2P*, 2003.

[2] R. Beverly. Reorganization in Network Regions for Optimality and Fairness. Master's thesis, MIT, Aug. 2004.

[3] Y. Chawathe, N. Lanham, S. Ratnasamy, S. Shenker, and L. Breslau. Making gnutella-like P2P systems scalable. In *Proceedings of ACM SIGCOMM*, 2003.

[4] B.-G. Chun, B. Y. Zhao, and J. D. Kubiatowicz. Impact of neighbor selection on performance and resilience of structured P2P networks. In *Proceedings of IPTPS*, Feb. 2005.

[5] S. T. Goh, P. Kalnis, S. Bakiras, and K.-L. Tan. Real datasets for file-sharing peer-to-peer systems. In *Proceedings of 10th International Conference of Database Systems for Advanced Applications*, 2005.

[6] K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proceedings of ACM SIGCOMM*, 2003.

[7] T. Joachims. Making large-scale SVM learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning.* MIT Press, 1999.

[8] T. Karagiannis, A. Broido, M. Faloutsos, and K. claffy. Transport layer identification of P2P traffic. In *Proceedings of ACM Sigcomm Internet Measurement Conference*, Oct. 2004.

[9] P. Keleher, B. Bhattacharjee, and B. Silaghi. Are virtualized overlay networks too much of a good thing? In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, Mar. 2002.

[10] A. H. Rasti, D. Stutzbach, and R. Rejaie. On the long-term evolution of the two-tier gnutella overlay. In *IEEE Golbal Internet*, 2006.

[11] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *Proceedings of INFOCOM*, 2003.

[12] D. Stutzbach, R. Rejaie, and S. Sen. Characterizing unstructured overlay topologies in modern P2P file-sharing systems. In *Proceedings of ACM Sigcomm Internet Measurement Conference*, Oct. 2005.

[13] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of ACM SIGCOMM*, 2003.

[14] V. N. Vapnik. *The nature of statistical learning theory.* Springer Verlag, 1995.

[15] Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval*, pages 42–49, 1999.

[16] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 412–420, 1997.