Application of Machine Learning to Link Prediction

Kyle Julian (kjulian3), Wayne Lu (waynelu)

December 16, 2016

1 Introduction (0.5 pages)

Real-world networks evolve over time as new nodes and links are added. Link prediction algorithms use historical data in order to predict the appearance of a new links in the network or to identify links which may exist but are not represented in the data. The application of link prediction is most commonly seen in recommendation engines, such as new connections on social networks or related products on shopping sites. Traditional approaches involve the calculation of a heuristic similarity score for a pair of nodes, such as the number of common neighbors or the shortest path length connecting the nodes, where pairs of nodes with the highest similarity scores are considered the most likely edges.

In this project, we will apply supervised learning algorithms to the link prediction prediction problem using a large set of topological features. Given a network at two different points in time, we train a learning algorithm to identify pairs of edges which appear in the newer network but not in the older network. Then, for a pair of nodes, we use the classification probability of the learning algorithm as our link prediction heuristic. Furthermore, we show that our network-specific heuristics outperform generic heuristics such as the Adamic/Adar coefficient and the Jaccard coefficient.

2 Related Work (0.5 pages)

Link prediction and the application of machine learning techniques to link prediction both have significant corpuses of work behind them. Adamic and Adar used similarities between the web pages of students at MIT and Stanford to predict friendship. Notably, in generating a similarity score, common features between students were weighted by the inverse log of their overall frequency [1]. Liben-Nowell and Kleinberg applied to idea of link predictors to network evolution by evaluating how well link prediction heuristics can account for the growth of a network over time. They used a variety of topological heuristic scores as link predictors for arXiv co-authorship networks and compared their relative performances [2]. In the same vein, Zhou et al. evaluated the performance of local similarity heuristics on six real-world networks from different domains and then described a new local similarity measure based on resource distribution through common neighbors [3].

Al Hasan et al. applied supervised learning to link prediction on the BIOBASE and DBPL co-authorship networks, using both topological and domain-specific proximity features [4]. Leskovec et al. applied supervised learning to edge sign detection in real-world social networks, notably comparing the performance of trained algorithms across different datasets and applying partial triads as features to capture the relationship of two nodes through their neighbors [5].

3 Datasets and Features (0.5 - 1 page)

The datasets used in this project are publicly available from the Stanford Network Analysis Project (SNAP) [6]. Our datasets consist of the Wikipedia Request for Adminship (RfA) voting network, five arXiv co-authorship networks, and two snapshots of the Slashdot Zoo social network. For each dataset graph $G = (V, E)$, examples were generating via the following process:

1. Remove low-degree nodes from G ($d_{\text{min}} = 5$ for Slashdot graphs, $d_{\text{min}} = 3$ for all others) to induce a subgraph $G_n = (V', E')$
2. Let E'' be a random sample of 10% of the edges of E'. Let $G_o = (V', E' - E'')$
3. Let $P \subset V' \times V'$ be all pairs of nodes (u, v) such that u and v share a common neighbor, but $(u, v) \notin E'$ and $u \neq v$
4. Apply a 70-30 training-testing split to E'' and P and extract the topological features using G_o.

Dataset	**Nodes**	**Edges**	**Training**	**Testing**
wiki-Vote	3772	98978	28491	66479
ca-AstroPh	14175	189929	56976	132944
ca-CondMat	14645	78785	23634	55146
ca-GrQc	2155	9967	2988	6972
ca-HepPh	7225	110243	33072	77168
ca-HepTh	4306	17306	5190	12110
slashdot0811	77360	905468	271638	633822
slashdot0902	82168	948464	284538	663922

Table 1: Summary statistics of the pruned datasets

The removal of a random sample of edges simulates the state of the graph at a previous point in time, G_o. Our positive training examples then consist of edges which do not appear in G_o but do appear in G_n, simulating the growth of the network. We balance our positive examples with negative examples which consist of pairs of nodes which do not form an edge in G_n. We then extract features from G_o because our goal to train a learner to predict edges in G_n using the current state of G_o.

For a pair of nodes (u, v), we extract the three different classes of features. There are eight degree features which are $d_{\text{in}}(u)$, $d_{\text{out}}(u)$, $d_{\text{in}}(u)/d_{\text{out}}(u)$, $d_{\text{out}}(u)/d_{\text{in}}(u)$, $d_{\text{in}}(v)$, $d_{\text{out}}(v)$, $d_{\text{in}}(v)/d_{\text{out}}(v)$, and $d_{\text{out}}(v)/d_{\text{in}}(v)$. There are eight triad features which are defined by the number of partial triads that u, v form with a common neighbor w. There are four types of triads with participation frequency for each denoted as t_1, t_2, t_3, and t_4. The eight triad features are: t_1, t_2, t_3, t_4, $t_1/C(u, v)$, $t_2/C(u, v)$, $t_3/C(u, v)$, and $t_4/C(u, v)$, where $C(u, v)$ is the total number of common neighbors. Finally, there are four heuristic features: common neighbors ($\Gamma(u) \cap \Gamma(v)$), the Adamic/Adar coefficient ($\sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(w)|}$), the Jaccard coefficient ($\frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|}$), and the preferential attachment coefficient ($|\Gamma(u)| \cdot |\Gamma(v)|$).

$u \to w \to v$	$u \to w \leftarrow v$	$u \leftarrow w \to v$	$u \leftarrow w \leftarrow v$
t_1	t_2	t_3	t_4

Table 2: The four triad types that u and v can form with a common neighbor w

4 Methods (1 - 1.5 pages)

We tested three different supervised learning algorithms for link prediction. The first algorithm, logistic regression, seeks to find θ to maximize

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{1}$$

This learner uses the logistic function, $g(z) = \frac{1}{1+\exp -z}$, which maps $\theta^T x$ to values between 0 and 1. Since we are training a binary classifier, the hypothesis represents the probability that the training label, y, is true. Using stochastic gradient ascent with the update rule,

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)})x_j^{(i)} \tag{2}$$

we can iteratively update θ to maximize the likelihood that the hypothesis is correct. Logistic regression is a standard algorithm for binary classification and serves as our baseline learner.

Our second learning algorithm is random forests which creates an ensemble of random trees and combines the individual decisions of the trees by taking the mode of the decisions. This algorithm draws random samples from the dataset with replacement and trains a decision tree on the sampled data. The decision tree is created by choosing splits that maximize the Gini impurity. The Gini impurity is a measure of disorder in a set by calculating the probability of mislabeling a random element in the set if it were labeled randomly. The split that decreases the Gini impurity the most is chosen as the decision split, and the tree is formed in a greedy fashion by sub-dividing the sets of training data into smaller and less disordered sets.

In many cases, decision trees can overfit to their training data too much. Some methods to improve this behavior include pruning the tree by stopping the decision splits where the Gini impurity falls below some threshold. The method chosen here to reduce over-fitting is to train multiple decision trees on random subsets of the data and combining the individual decisions into one decision. Each individual tree may ovefit to its random training data, but in general the forest of decision trees will be less biased to the draw samples.

Our last supervised learning method trains a neural network to classify the edges. For a binomial classification network, the output of the network will have two outputs that go through a softmax activation function:

$$h(z)_j = \frac{e^{z_j}}{e^{z_0} + e^{z_1}} \; j \in \{0, 1\} \tag{3}$$

The softmax converts the network outputs into a probability distribution in which the larger valued output will have a higher probability. The training labels can be represented as the true label probabilities, which take the form of one-hot vectors. We can compare the probability distributions using the cross-entropy method, which is a measure of difference between two probability distributions:

$$L(p, q) = -\sum_x p(x) \log q(x) \tag{4}$$

We can use stochastic gradient descent with cross-entropy loss to iteratively update the network parameters to reduce the loss as much as possible. The result is a network that produces accurate probability estimates of the data labels.

5 Results and Discussion (1-3 pages)

We implemented the three learning algorithms in Scikit-learn, a python library for machine learning. We did not normalize the training data to zero the mean and have unit variance for each feature. We wanted to create models that predict edges accurately even on datasets with different means and variances, and normalizing the datasets inhibits cross-prediction testing. To reduce overfitting, the logistic regression and neural network use L2 regularization, which adds a penalty to the cost function for having larger weights. Logistic regression was optimized using limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS), an approximate quasi-Newton method [7].

The random forest generates 11 random trees and draws the same number of samples as are present in the data set, but uses replacement to create distributions that differ.

The neural network has 5 hidden layers of 30 units each. Testing a few datasets with different network architectures showed that this shape gave the best performance without increasing the size of the network too much. Adam was used as the network optimizer, which estimates lower order moments of the loss in order to minimize loss very quickly without the need for finely tuning other parameters such as learning rate and momentum [8].

We trained each learner on three different subsets of the features to better understand their value in learning. The first subset uses only the first eight features, which look at the degrees of the source and destination nodes. The second subset includes the basic features as well as the triad features. The last subset also adds the heuristics as features. The learners were compared to the Adamic/Adar heuristic as well.

To compute test accuracies, we want to measure how accurately the learner predicts links. Given the test set for each dataset, which includes edges not in G_o but present in G_n, we use the learner to predict the probabilities that each node pair contains a link. If k links appear in G_n in the test set, then we take k none pairs with the highest probability of having a link and compute the percent of those pairs that have a link, as shown in Section 5.

The results show poor performance with only the basic features, which is expected given the features contain no information about the closeness of the two nodes. Including the triads, which has information about the proximity of the nodes, greatly improves the learners performance and surpasses Adamic/Adar. Including the heuristic features helps random forests the most since the heuristics are made to function like a decision boundary, making them easy to integrate into the decision trees while the other learners to not benefit much from the heuristic features.

We also tested random forest models on each dataset to investigate cross-prediction performance. The results show that datasets test well on models trained on similar data sets, but the learners cannot generalize well for every graph. The features given vary drastically between the datasets, so features that may indicate a link may not be good indicators in a different graph. However, we can see strong correlation among the co-authorship graphs and the slashdot datasets. This shows a learner trained can generalize well within a specific type of graph.

6 Conclusion and Future Work (1-3 paragraphs)

Through supervised learning, we are able to train learning algorithms to calculate network-specific similarity scores which outperform traditional similarity heuristics. Notably, the use of only degree and triad features is enough to have higher accuracy than our baseline of the Adamic/Adar coefficient. Because our features depend on the topological structure of the network and do not

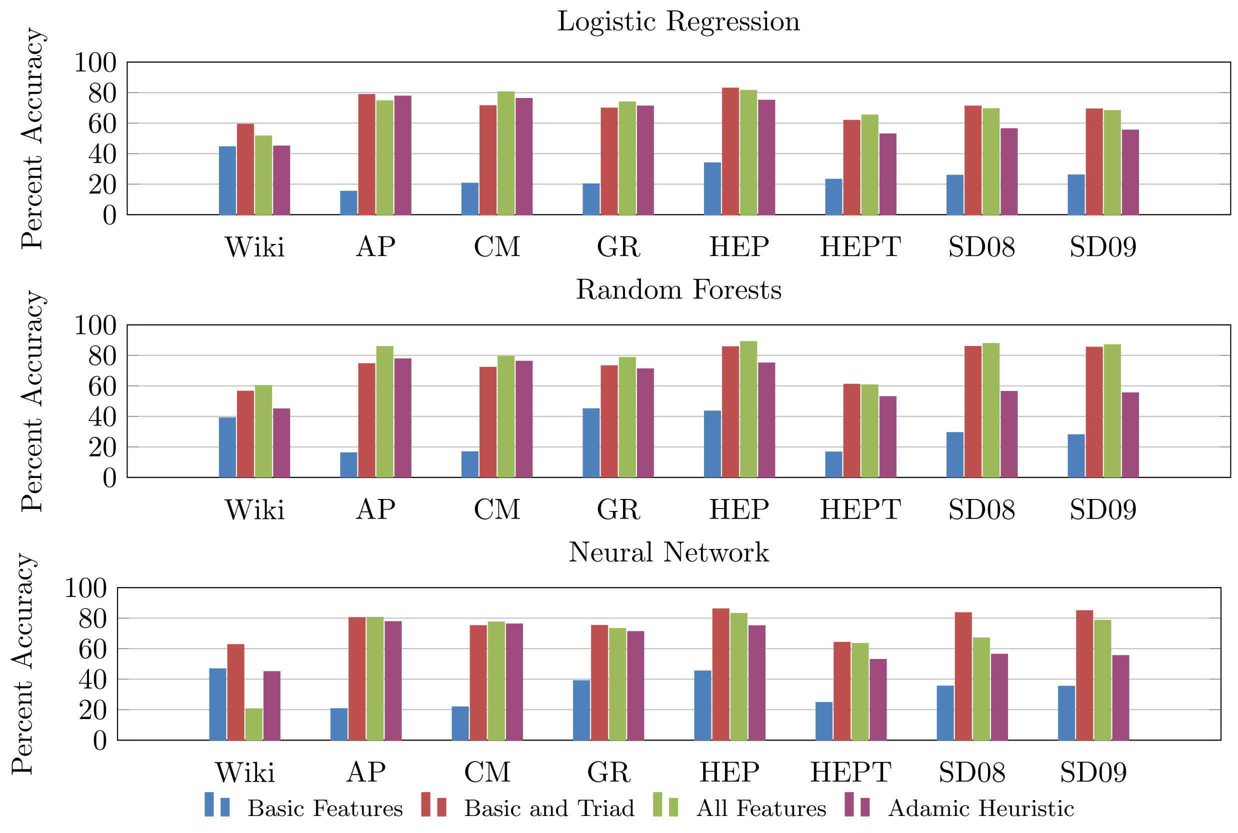

Figure 1: Accuracy of Link Predictions

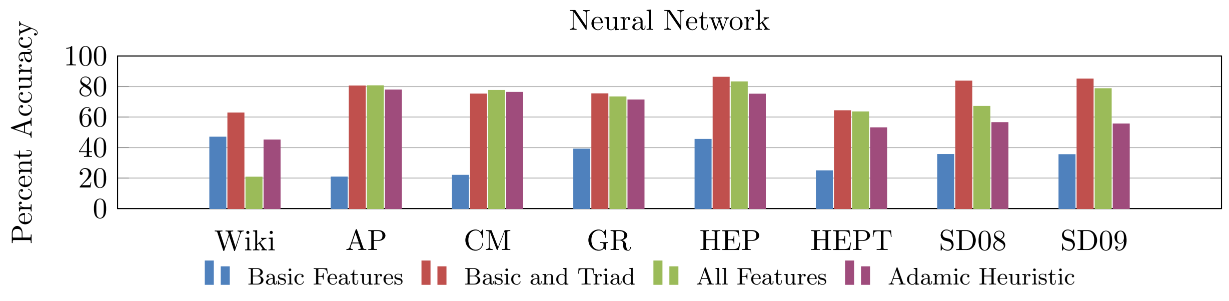

Figure 2: Accuracy of Cross Predictions

use domain-specific features, it is possible to investigate similarities between network structures by cross-evaluating the performance of the learning algorithms.

It should be noted that our work roughly simulated network growth by randomly removing edges. Unfortunately, organic real-world network growth is not random but tends to follow a preferential attachment model. Therefore, to truly draw any conclusions, our methods should be used on real network growth data by capturing a network at two distinct times. Further evaluation of the generalizability of our methods across different domains is also warranted. In our results, we were able to show that networks drawn from different domains exhibited different network structures which degraded the performance of link predictors. However, we did not have the time to collect and process additional datasets to find networks from different domains which showed similar cross-performance.

7 References

1. Adamic, Lada A., and Eytan Adar. "Friends and neighbors on the web." *Social networks* 25.3 (2003): 211-230.

2. Liben-Nowell, David, and Jon Kleinberg. "The link-prediction problem for social networks." *Journal of the American society for information science and technology* 58.7 (2007): 1019-1031.

3. Zhou, Tao, Linyuan L, and Yi-Cheng Zhang. "Predicting missing links via local information." *The European Physical Journal* B 71.4 (2009): 623-630.

4. Al Hasan, Mohammad, et al. "Link prediction using supervised learning." *SDM06: workshop on link analysis, counter-terrorism and security.* 2006.

5. Leskovec, Jure, Daniel Huttenlocher, and Jon Kleinberg. "Predicting positive and negative links in online social networks." *Proceedings of the 19th international conference on World wide web. ACM*, 2010.

6. Leskovec, Jure and Andrej Krevl. "SNAP Datasets: Stanford Large Network Dataset Collection." http://snap.stanford.edu/data. Accessed December 2016.

7. Zhu, Ciyou, et al. "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization." *ACM Transactions on Mathematical Software (TOMS)* 23.4 (1997): 550-560.

8. Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).