Improving Recall and Precision in Graph Convolutional Networks for Node Classification using Node2Vec Embeddings

Julio A Martinez
CS224W Final Report
Stanford University
`julioamz@stanford.edu`

Abstract

Graph Neural Networks (GCNs) have received a significant amount of attention recently because of their ability to generalize neural networks to graphs. They have shown promise in node classification by demonstrating the ability to have high accuracy. This works seeks to show that while GCNs inherently learn embeddings from the graph structure to make classifications, their recall and precision performance can still be improved using unsupervised feature extraction including Node2Vec and Egonet features with Rolx and Refex. In this work, I used an email dataset and extract features for node embeddings using unsupervised learning. These feature vectors are then fed into a GCN and demonstrate an improvement in recall and precision despite a small improvement in accuracy. Thus it is important to note that while unsupervised embeddings may not significantly improve accuracy, they provide more robustness in recall and precision.

1 Introduction

Communities often arise in networks, ranging from social networks, biological networks to technological as well as shopping networks. However, detecting these densely linked groups is challenging since the notion of a community may vary from one network to another. Fortunately there are many data sets for which we have ground-truth communities as in [1] which we can test our ability to test our methods on identifying a community or testing the score of the goodness of a community.

The problem as mentioned above is that communities may come in different forms. Take for instance a community based on publications, where a publication can be the node in a graph and the type of publication is the the topic. In this case, publications from different topics may be links as well as publications from similar topics (assume that a link is simply a citation and ignore directed networks). However, a different network community may be individuals sending emails, where a link between two individuals exists if these two individuals have exchange an email, and not if these two individuals have not. The type of the node in this network may be the institution or organization the individual may come from. You can see the in both of these examples, the networks may exhibit very different properties. It is possible that in an email network it is very common for individuals to email people across an organization instead of only within a group or team due to difficulty of getting in contact. On the other hand the publications community may be such that there is a strong bias towards citing individuals from the same research areas. Thus defining community strictly as a strongly linked set of entities may not be the best answer for every network. Hence we need methods that are able to be robust enough to predict the community a node belongs to by using node classification.

2 Problem Statement

This work is interested in understanding the robustness of classifying a node's community or ground truth label. This project seeks to predict the class a node belongs to using neural networks. Communities can be thought of as the nodes which correspond to some specific label. Thus all nodes with the same label belong to the same community. Since different networks may exhibit different properties with respect to their communities, machine learning is a great candidate for learning these properties to classify test nodes. This work uses Graph Convolutional Networks (GCNs) [5] to learn classifiers that can predict the class of a node. Success is measured by computing recall, precision, and accuracy of predictions on a test set and they are compared to a base case using logistic regression.

3 Related Work

In [5] Graph Convolutional Networks are used for classifying nodes. However in this work features vectors inherent from the data are used instead of using features extraction. If no feature vectors are available, features are assumed to be the identity matrix. When relying only on an identity feature matrix the GCN to learns the graph structure on its own without drawing any correlations from the features since there are none. However, in this work we will see that although manual extraction of feature vectors may not significantly improve accuracy for a trained GCN model, extracting features may lead to improvement in recall and precision.

There are many forms of feature extraction for local structure of nodes. In [3], Rolx is introduced. Rolx is a method for extracting local node structural features. In addition to Rolx, Refex is introduced in [4] where recursive calls to Rolx are made to introduce features that include more information as from further away the node as each recursive call is made. Although the dimensionality of a Rolx feature vector is known to grow exponentially, at each recursive call, it provides with each call a greater scope or understanding of the structure around a node. More recursive calls include more information about the graph around the node, however they require significantly larger feature vectors that may be too large for practical use cases.

Node2Vec in [6] is a framework that learns embeddings for nodes in a graph. These embeddings are learned in an unsupervised manner, by exploring the graph. This is done by simulating random walks starting from each node. The resulting node embedding captures and represents the connectivity of that node to the other nodes in the graph, thus representing the relationship it has with other nodes.

Given a node u in a graph, a random walk is simulated with some fixed length. If we let c_i represent the i'th node along the random walk then $c_0 = u$. The result are nodes c_i which help define the probability distribution of going from any node to the next:

$$P(c_i = z | c_{i-1} = v) \tag{1}$$

This probability is equal to $alpha_{pq}(t, z)w_{vz}$ where v and z are nodes in the graph and w_{vz} is simply the edge weight (in this work all edges weights are 1). The way the random walk is carried out is controlled by two parameters p and q which define the tendency of the walk to go away or stay within the same distance or come closer to the starting node u. Hence p is known as the return parameter, smaller p will make the random walk more likely to return and q is known as the in-out parameter where low q will make the random walk go further out, and small q will keep the random walk in close.

4 Data

The dataset used in this project is from [2] which is known as the email-Eu-core dataset. This dataset contains data from email exchanges from a large European institution. This is a directed graph where an edge exists between two members of the institution if member u has sent an email to member v. However for the purposes of this project, this is treated as undirected, since all we care about is to know what department (class) does an individual (node) belong to. For each individual in the institution, the ground-truth community membership is given and there are 42 communities or departments to which an individual can belong to.

The graph contains 1005 nodes, 16,706 edges, and 42 ground truth labels. Taking a look at figure 1 we can see that this network of emails shows a typical log distribution.

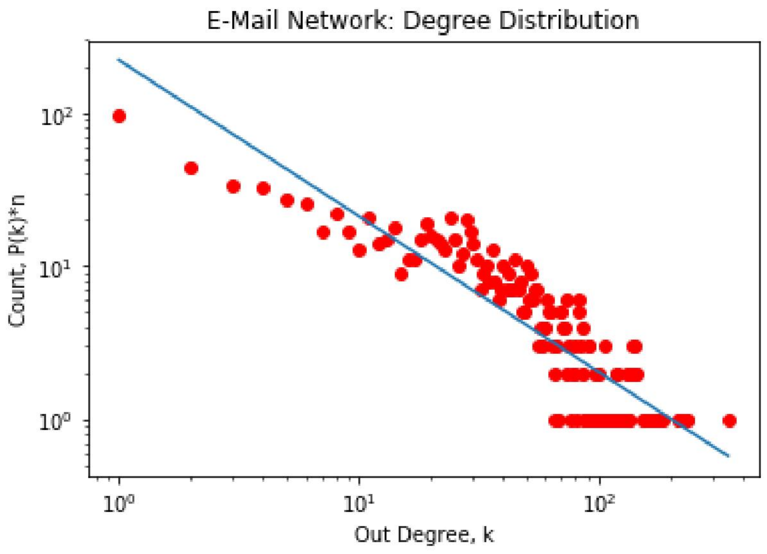

Figure 1: Node Degree Distribution

One thing however, that is important to keep in mind which will come up later in the results is that the dataset has a significant imbalance representation. In figure 2 we can see that nodes from class 5 (some department) has a very large representation of over 100 emails. This means this department is very active, sending emails, while other departments, such as department 19 have only 1 email on record. Because of this imbalance and low representation of some departments, it will be impossible to to a data split which contains a nodes for every department in the validation and test sets. Therefore, I opted to keep the lower represented nodes in the training set, although, our test sets will never see them, and it will still be possible to predict them.

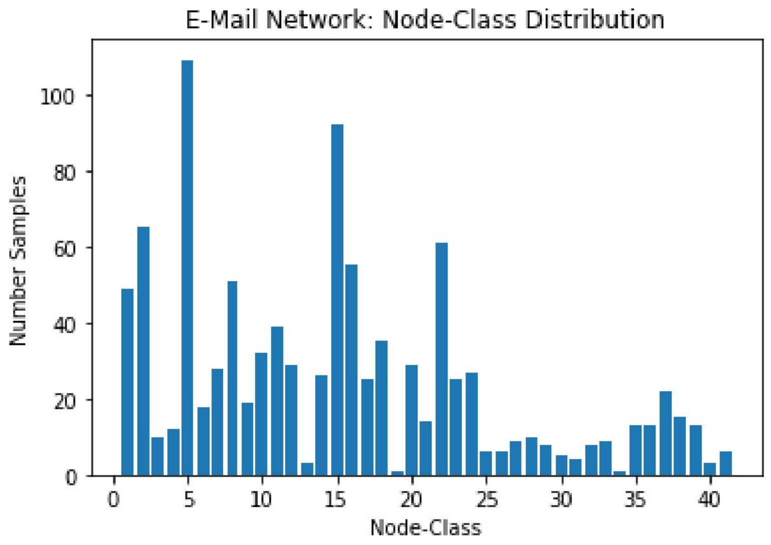

Figure 2: Imbalance in Department Representation

The data split for the email network is done by shuffling the nodes and taking 80% as the training set, 10% as the validation set, and the remaining 10% as the test set.

5 Method and Approach

I used a combination of of techniques in my approach to classification. This section will talk about these techniques and and how I used and implemented them.

5.1 Rolx and Refex

In order to prepare the dataset to be used by a GCN I first extract the local node features for each node across the entire network. This is done with the algorithms Rolx from [3] and ReFex in [4] for structural role and recursive feature extraction respectively. However, unlike the homework where we simply use the features extracted from the two algorithms, we used them as feature inputs into the GCN.

As is very well known the number of features using the Rolx and Refex method increases exponentially upon each recursive call. The trials included in this milestone vary from 0 recursive calls (i.e. just using Rolx alone) which gives three features all the way up to 6 recursive Refex calls which results in $3^(6 + 1) = 2,187$ features. What is clear is as the number of features grows the more information we have on a particular nodes relationship to the rest of the graph, however, what we will see is that there is a significant trade off between testing accuracy and the number of features.

$$V'(u) = [V(u), \frac{1}{N(u)} \sum_{v \in N(u)} V(u), \sum_{v \in N(u)} V(u)] \tag{2}$$

5.2 Node2Vec

For Node2Vec I ran a number of trials using 80 steps as the length of the walk, a dimension of 128 for the embedding, 10 as the number of walks, and experimented with 5 different sets of (p, q) pairs. The different pairs were $(1, 0.1)$, $(0.1, 1)$, $(0.5, 0.5)$, $(0.3, 0.8)$, $(0.8, 0.3)$. These embeddings were done with the implementation in [8]

5.3 Logistic Regression Baseline

A multinomial logistic regression model was used as my baseline method. The feature representations of the nodes, both the Egonet feature vectors and the Node2vec embeddings, were used as the feature inputs to train the logistic regression models. This used the same data split mentioned in the data section and which was used for the GCN model in this work. All classes were treated equally and hence the imbalance in the data set as discussed previously will play a role in classification accuracy, precision and recall. The classification task is to classify the node's label (department) into one of the 42 labels it can possible belong to.

5.4 GCN

GCNs as shown in [5] are simply a class of neural networks that generalize the concept of neural networks to graphs. What is useful here is that GCNs learn a mapping from the input node features X (which we can assume is an NxD matrix for N nodes where each node is associated with a D dimensional feature vector) and the input adjacency matrix A which is the adjacency matrix for a graph $G = (V, E)$ to the output Z (in the case of node embedding a NxF matrix, i.e. N nodes each of which has an F dimensional node embedding).

A given layer l for a Graph Neural Network is then as follows:

$$f(H^{(l)}, A) = \sigma(D^{-1/2}(A + I)D^{-1/2}H^{(l)}W^{(l)}) \tag{3}$$

I is the identity matrix which is added to the adjacency matrix to avoid disregarding the node in question (instead of only aggregating across the node's neighbors). D is simply the diagonal node degree matrix (can be computed by summing the across the rows of A. Thus D acts as a symmetric normalizer of the adjacency matrix A. Finally σ denotes as usual the activation function to be used. In my work the activation function used is a ReLU activation.

The complete topology of the network implemented in this work was implemented by starting with the starter code in [9] and adapting to the following forward pass architecture shown below:

$$\tilde{A} = A + I_N \tag{4}$$

$$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij} \tag{5}$$

$$\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2} \tag{6}$$

$$Z^{(1)} = ReLU(\hat{A}XW^{(1)} \tag{7}$$

$$Z^{(2)} = Softmax(\hat{A}Z^{(1)}W^{(2)}) \tag{8}$$

Important to keep in note is that X is the feature vector that I feed in. This may either be the identity matrix or the Egonet or Node2Vec embeddings. This entire feature vector is fed (all train, validation, and test) into the network, along with the adjacency matrix A. The loss however, uses a mask to only compute the loss over the training examples. This way we get the added benefit of the entire structure of the graph, which is not an option in Logistic Regression relying solely on the node features. The loss is simply a Cross Entropy loss function which we wish to minimize over using Adam gradient descent.

As shown in [5], the first step is to initialize the weights W randomly and X to simply be an identity matrix. In my tests, I run Xavier initialization. In the first step, the forward pass performs convolution on the graph, or convolves the 3-rd order neighborhood of every node. This simply means that weights W act as a filter that convolve on the neighbor hood of nodes within 3 hops away from a any node. During this process the the model produces an embedding. This embedding captures the community structure of the graph. In fact, this comes at no cost of back propagation yet. Finally we can start training by updating the weights, this is the part where the model is expected to linearly separate the communities found in the graph.

6 Results and Findings

The base case was to use logistic regression. Metrics used to measure success are recall, precision and accuracy. However, since the dataset is imbalanced as shown previously, a reweighed recall and precision are included in order to weigh the recall and precision proportionally by the number of samples available for each class or label.

The first test was using Node2Vec alone. Running initial test on a GCN before tuning hyperparameters, it was clear that the GCN models could only take advantage of the Egonet features until they became too large to be useful. In 3, you can see the progression of accuracy increase and then decrease. This is due to the exponential growth of the feature vector size having high dimensionality for the dataset size of the email network.

The first test case was to try logistic regression with the Egonet features and the Node2vec features. The Egonet features used in the provided results is after 5 iterations of a recursive call to Refex as described in the methods section. As is clear from the logistic regression table, Node2vec has clear advantages in every metric. The features leading to best performing logistic regression is Node2vec with parameters $p = 0.1$ and $q = 1$, which keeps the random walk returning to previous nodes and close in diameter. This may be indicative of the way emails are sent at the

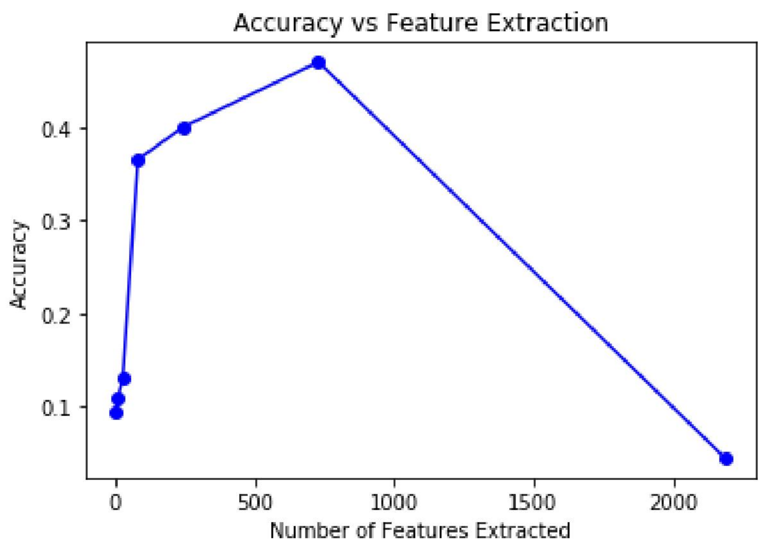

Figure 3: Accuracy vs vs Number of Egonet Features on email data

institution, where emails stay within tightly connected departments instead of across departments.

Logistic Regression					
Model	Recall	W-Recall	Precision	W-Precision	Accuracy
LR-Egonet	0.2	0.18	0.4	0.32	0.4
LR-Node2Vec, p=0.1,q=1	0.68	0.61	0.82	0.77	0.80
LR-Node2Vec, p=1,q=0.1	0.57	0.52	0.75	0.72	0.75
LR-Node2Vec, p=0.5,q=0.5	0.64	0.59	0.78	0.73	0.78
LR-Node2Vec, p=0.3,q=0.8	0.66	0.58	0.78	0.75	0.78
LR-Node2Vec, p=0.8,q=0.3	0.59	0.53	0.77	0.74	0.77

The GCN models were trained using training set and validation set to tune the hyperparameters. The training curve along the validation and test curve are shown below in figure 4

After running the GCN models we are able to again see similar behavior as shown in the GCN table. The best peforming GCN has accuracy 0.81 and uses node2vec feature embeddings with $p = 0.1$ and $q = 1$ (as with Logistic Regression). Although the improvement is only slight with respect to accuracy, it is quite clear that GCNs have improvement in recall and precision on almost all cases with one minor exception, the Identity input case. Here we see that the identity feature input case still manages to get 0.74 in accuracy, however, performs much lower in recall, which is the number of relevant items actually predicted correctly. We can see that once we introduce Node2Vec embeddings and even Egonet embeddings, the GCN is

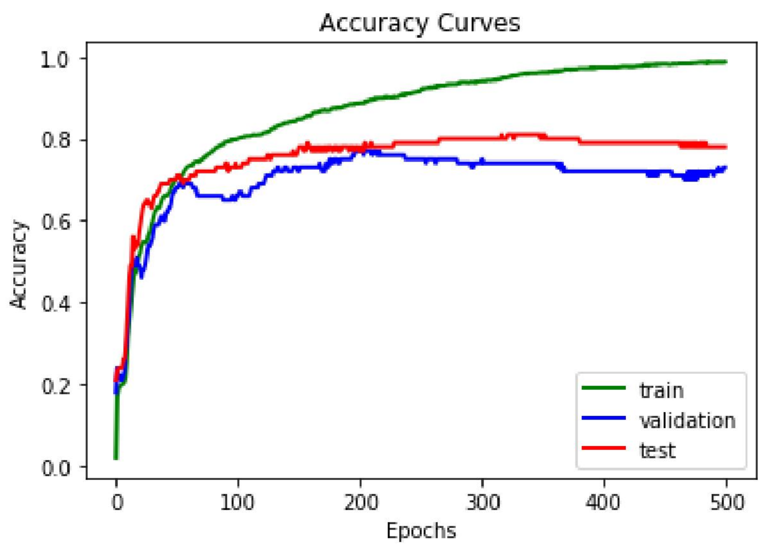

Figure 4: Training, Validation, and Test Curves for Accuracy

able quite uniformly improve recall and even precision for all cases except for the Identity.

Graph Convolutional Network					
Model	Recall	W-Recall	Precision	W-Precision	Accuracy
GCN-Egonet	0.59	0.56	0.80	0.78	0.8
GCN-Identity,	0.51	0.5	0.74	0.76	0.74
GCN-Node2Vec, p=0.1,q=1	0.69	0.66	0.81	0.83	0.81
GCN-Node2Vec, p=1,q=0.1	0.62	0.63	0.76	0.76	0.76
GCN-Node2Vec, p=0.5,q=0.5	0.64	0.65	0.78	0.78	0.78
GCN-Node2Vec, p=0.3,q=0.8	0.69	0.65	0.78	0.76	0.78
GCN-Node2Vec, p=0.8,q=0.3	0.67	0.69	0.77	0.84	0.77

Taking a look at a projection of the Egonet embeddings of two departments, we get the following as shown in figure 5.

In contract, we can compare these to the embeddings from the Node2vec projections in **??**. It is clear from the projections that the Node2vec embeddings allows for a much more separable data set. While there still exists cases that are difficult, especially due to low amount of data, Node2vec is clearly much more power than Egonet and identity when used by GCN since is maintains accuracy and increases recall and precision.

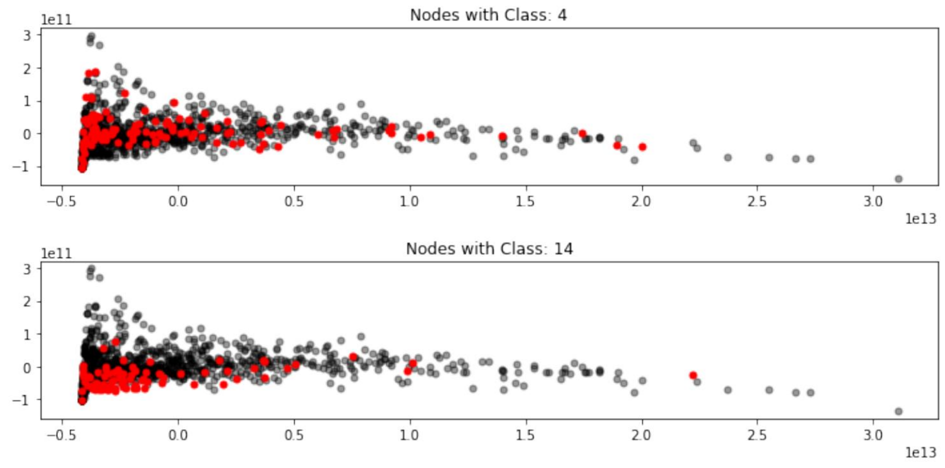

Figure 5: Egonet tsne Projections

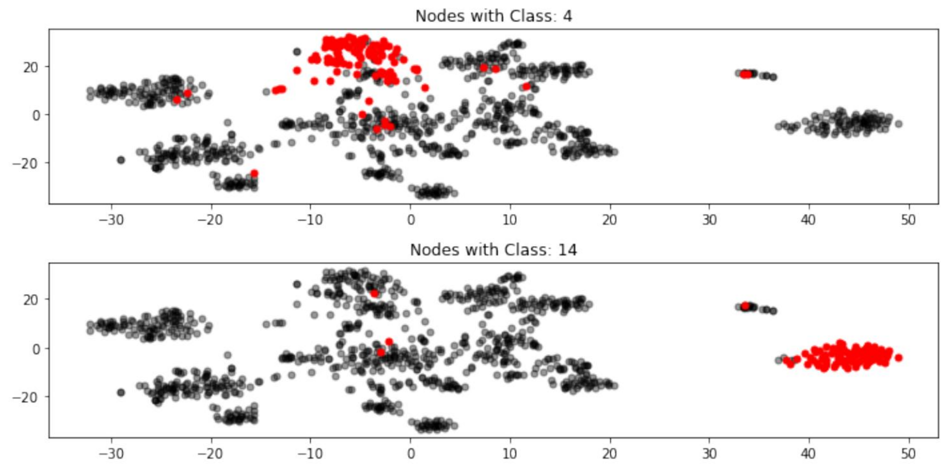

Figure 6: Node2Vec tsne Projections

7 Discussion, Conclusion, and Future Steps

It is clear that Node2Vec and Egonet Features provide the GCN with improvements in precision and recall. One logical explanation for this is that these features provide more information than the GCN is able to learn on its own. The GCN is limited by the number of layers it has, in this work 2, and therefore the number of hops at which it can learn for the structure of the graph is limited. Node2Vec and Egonet easily provide the GCN to latch onto information that was not available previously without adding more layers to the network. Although is has not been shown, adding more layers may lead to over fitting since the dataset is small to being with.

In conclusion, it is clear adding feature embeddings allow the network to be more robust and achieve higher recall and precision. Some ideas on future work include verifying this empirically on a variety of datasets. Adding layers to the GCN to see if the GCN is able to improve in recall and precision without the embeddings fed into it. Finally, trying different dimensions for embeddings to see the trade off on dimension vs recall and precision.

References

[1] Yang, Jaewon, and Jure Leskovec. "Defining and evaluating network communities based on ground-truth." Knowledge and Information Systems 42.1 (2015): 181-213.

[2] Yin, Hao, et al. "Local higher-order graph clustering." Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2017.

[3] Henderson, K., et al. RolX: Role Extraction and Mining in Large Networks. No. LLNL-TR-498952. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2011.

[4] Henderson, Keith, et al. "It's who you know: graph mining using recursive structural features." Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2011.

[5] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional net- works." arXiv preprint arXiv:1609.02907 (2016).

[6] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2016.

[7] Hamilton, William L., Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs." arXiv preprint arXiv:1706.02216 (2017).

[8] https://github.com/aditya-grover/node2vec

[9] http://snap.stanford.edu/deepnetbio-ismb/ipynb/Graph+Convolutional+Prediction+of+Protein+Interactions+in+Ye

[10] Final Project Source Code github.com link