2017 IEEE 33rd International Conference on Data Engineering

# On Edge Classification in Networks with Structure and Content

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
Email: charu@us.ibm.com

Yao Li, Philip S. Yu
University of Illinois at Chicago
Chicago, IL 60607
Email: {yli70, psyu}@cs.uic.edu

Yuchen Zhao
AppDynamics, Inc.
San Francisco, CA 94107
Email: zycemail@gmail.com

***Abstract*—The problem of node classification has been widely studied in a variety of network-based scenarios. In this paper, we will study the more challenging scenario in which some of the *edges* in a *content-based* network are labeled, and it is desirable to use this information in order to determine the labels of other arbitrary edges. Furthermore, each edge is associated with text content, which may correspond to either communication or relationship information between the different nodes. Such a problem often arises in the context of many social or communication networks in which edges are associated with communication between different nodes, and the text is associated with the content of the communication. This situation can also arise in many online social networks such as chat messengers or email networks, where the edges in the network may also correspond to the actual content of the chats or emails. The problem of edge classification is much more challenging from a scalability point of view, because the number of edges is typically significantly larger than the number of nodes in the network. In this paper, we will design a holistic classification approach which can combine content and structure for effective edge classification.**

## I. Introduction

In edge classification of networks, labels are associated with a subset of the edges and it is desired to classify the unlabeled edges. This problem has been studied recently in the context of binary prediction in signed networks [2], and in the generic multi-class setting [1]. The problem of edge classification of social networks with content arises in a number of different scenarios. In counter-terrorism applications, it may be desirable to monitor the communications between entities in order to detect suspicious communications. In such cases, both the content and the structure of the communication can play a role in the classification process. In many social networking applications, numerous forms of communications between entities (such as wall posts in *Facebook*) are public in nature. The content and local structure of such posts can be used in order to classify its relevance to a particular product.

The most straightforward approach is to directly use text classifiers in order to perform the classification. However, such an approach ignores the rich structural information which is often available in the context of a network. A second way to perform the classification is by using the information which is latent in the underlying link structure. However, most of the existing methods are designed for *node* classification rather than *edge* classification. In recent years, a number of methods have been proposed for binary class (sign) prediction [2] and multiway edge-label prediction, although these methods are designed only for structural settings without the use of content.

It is clear that both text and links encode important information about the underlying network. Furthermore, these provide different views of the underlying information. For example, the text provides ideas about content, whereas the linkage behavior provides information about interconnectedness between different kinds of nodes, some of which may be used for classification. In some cases, the content at the edges may be very limited, as a result of which the structure may provide more useful hints about the classification process. In other cases, where the edge is in a sparsely labeled region of the network, the content may provide the only useful information for classification purposes. In this paper, we will design a scalable matrix-factorization approach for classification. We will refer to this algorithm as *ECONOMIX*, which corresponds to the fact that it is a Edge and Content based classification algorithm with Matrix Factorization.

## II. Basic Edge Classification Algorithm

We assume that we have a network $G = (N, A)$ with node set $N$ and edge set $A$. We assume that the number of nodes in $N$ is denoted by $n$. The edge set $A$ contains the set of $m$ edges denoted by $\{e_1 \dots e_m\}$. We assume that the nodes associated with the $j$th edge are denoted by $(s(j), t(j))$, where both $s(j)$ and $t(j)$ are drawn from the node set $N$. We note that the values of $(s(j_1), t(j_1))$ and $(s(j_2), t(j_2))$ may be identical for two different indices $j_1$ and $j_2$. This corresponds to parallel edges which correspond to multiple communications between the same pair of entities. The edge $e_j$ has text content associated with it, which is denoted by the document $D(j)$. The value of $D(j)$ will typically be different across parallel edges between the same pair of entities. Some of the edges also have class labels associated with them. We assume that the label associated with the $j$th edge is denoted by $l(j) \in \{1, 2, \dots k\}$. In the event that there is no label associated with the $j$th edge, then we assume that $l(j)$ takes on a null value, which are the candidates for classification.

We will first describe a simple algorithm for edge-classification which uses matrix factorization. The first step in the classification process is to prune the lexicon, so that the discriminative words are used in the classification process with the use of the Gini index. Let $p_i(w)$ denote the fraction of documents containing word $w$, which also belong to label $i$. In other words, $p_i(w) = P(l(j) = i | w \in D(j))$. Then, the (inverted) Gini index $G(w)$ of the word $w$ is defined as $G(w) = \sum_{i=1}^{k} p_i(w)^2$. The value of $G(w)$ lies between $1/k$ and 1, and is larger for more discriminative words. We determine the mean $\mu$ and standard deviation $\sigma$ of $G(w)$ over the different words, and retain only those words for which the $G(w)$ is at least $\mu - 0.5 \cdot \sigma$. Let $d$ be the total number of words remaining in the lexicon, once the pruning has been performed.

In order to perform the matrix factorization, we utilize two matrices denoted by $T$ and $L$. The matrix $T$ denotes the $m \times d$ document-term matrix, in which the $(i,j)$th entry corresponds to the fractional frequency of the $j$th term in the $i$th document $D(i)$ for edge $e_i$. The matrix $L$ is node-edge matrix of size $m \times n$. In the matrix, the $i$th row corresponds to the edge $e(i)$, and the matrix contains one column for each node. Exactly two entries in each row take on the value of 1, and these are the edges which are the end points of that node. All other entries in the row take on the value of 0.

Then, for some balancing parameter $\lambda$, we define the $m \times (n+d)$ matrix $S(\lambda)$, as the row-wise concatenation of the matrix $T$ with the matrix $\lambda \cdot L$. In other words, each row of $S(\lambda)$ is the concatenation of the corresponding rows of $T$ and $\lambda \cdot L$. Therefore, we have $S(\lambda) = [T, \lambda \cdot L]$. We note that $\lambda$ is a balancing parameter that regulates the relative importance of link and content in the classification process. We will discuss the precise details of picking $\lambda$ later.

In order to perform the matrix factorization, we first need to fix the number of clusters found by the algorithm. Let $q$ denote this number of clusters. The value of $q$ then regulates the dimensions of the matrices into which the factorization is performed. We perform the factorization into a $m \times q$ matrix $U$, and a $(d+n) \times q$ matrix $V$ as $S(\lambda) \approx U(\lambda) \cdot V(\lambda)^T$.

The matrices $U(\lambda)$ and $V(\lambda)$ may be found by setting up the optimization problem which minimizes $J = ||S(\lambda) - U(\lambda) \cdot V(\lambda)||$. Here $||\cdot||$ corresponds to the sum of the squares of all the entries in the resulting matrix. This value needs to be optimized subject to the constraints that the matrices $U(\lambda)$ and $V(\lambda)$ need to be non-negative. We can use the standard Lagrangian optimization method in order to set up the iterative update equations of this method. Let $U(\lambda) = [u_{ij}(\lambda)]$ and $V = [v_{ij}(\lambda)]$ be the parameters of the factorization matrices.

This leads to the following iterative update rules for $u_{ij}(\lambda)$ and $v_{ij}(\lambda)$ (which is well-known in matrix factorization):

$$u_{ij}(\lambda) = \frac{(S(\lambda) \cdot V(\lambda))_{ij} \cdot u_{ij}(\lambda)}{(U(\lambda) \cdot V^T(\lambda) \cdot V(\lambda))_{ij}}$$

$$v_{ij}(\lambda) = \frac{(S^T(\lambda) \cdot U(\lambda))_{ij} \cdot v_{ij}(\lambda)}{(V(\lambda) \cdot U^T(\lambda) \cdot U(\lambda))_{ij}}$$

We note that the rows of $V(\lambda)$ correspond to the latent factors of the matrix $S(\lambda)$ both in terms of its content and links. Each row of $V(\lambda)$ has a length of $(d+n)$, and the values represent the importance of the $d$ different words, and $n$ different nodes in the cluster. Each row of $U(\lambda)$ corresponds to a coordinate representation of an edge in terms of these latent factors. Specifically, the $i$th row of $U(\lambda)$ corresponds to the $i$th edge $e(i)$. The index of the largest among the $q$ entries (coordinates) in the $i$th row provides the cluster index for that edge. Thus, for each edge, we can define a corresponding cluster index with the use of matrix factorization.

```
Algorithm FindContentLinkCombo(Doc. Matrix: $T$, Link Matrix: $L$);
begin
  Set $\lambda = \lambda_0 = 1$; $\epsilon = 0.1$;
  Compute $Q(\lambda)$, $Q(\lambda + \epsilon)$ and $Q(\lambda - \epsilon)$;
  if $Q(\lambda)$ is largest then return($\lambda$);
  if $Q(\lambda + \epsilon)$ is largest then
  begin
    Determine $Q(\lambda + \epsilon)$, $Q(\lambda + 2 \cdot \epsilon)$ ... $Q(\lambda + r \cdot \epsilon)$ until
     the value no longer increases;
    $\lambda' = \lambda + r \cdot \epsilon$;
    Perform binary search in $[\lambda_0, \lambda']$ determine optimal $\lambda$;
    return optimal value of $\lambda$;
  end
  if $Q(\lambda - \epsilon)$ is largest then
  begin
    Determine $Q(\lambda - \epsilon)$, $Q(\lambda - 2 \cdot \epsilon)$ ... $Q(\lambda - r \cdot \epsilon)$ until
       the value no longer increases;
    $\lambda' = \lambda - r \cdot \epsilon$;
    Perform binary search in $[\lambda', \lambda_0]$ to determine optimal $\lambda$;
    return optimal value of $\lambda$;
  end
end
```

Fig. 1. Determining an Appropriate Combination of Content and Links

We use this clustering in order to determine the cluster purity with respect to the class labels. The dominant class label of a cluster is defined as the fraction of the points in the cluster, which belong to the class label with the largest presence. Specifically, for each of the $i \in \{1 \dots q\}$ clusters, let $n(i)$ be the number of points in the cluster, and let $\alpha_i$ be the fraction of the cluster which belong to the dominant class label of cluster $i$. Then, the overall class label purity $P$ is defined a s the (weighted) average of the values of $\alpha_i$ over the different clusters. Therefore we have $P = \sum_{i=1}^{q} n(i) \cdot \alpha_i / \sum_{i=1}^{q} n_i$.

Since we will implicitly use the clustering in order to create a locality-based classifier, we would ideally like to create a clustering which has as high a cluster purity as possible. Let us denote the afore-mentioned value of the purity $P$ for a particular value of $\lambda$ by $Q(\lambda)$. Recall that the choice of $\lambda$ regulates the relative importance of structure and content in the classification process.

We use an iterative search method in order to determine the optimal value of the parameter $\lambda$ for the clustering process. We start off by picking $\lambda^0 = 1$ and determine the values of $Q(\lambda+\epsilon)$ and $Q(\lambda-\epsilon)$ for some small value of $\epsilon$ such as 0.1. In the event that $Q(\lambda)$ is larger than both $Q(\lambda+\epsilon)$ and $Q(\lambda-\epsilon)$, then we terminate the algorithm, and use the current value of $\lambda$ as the optimal value. Otherwise, we determine the largest among $Q(\lambda + \epsilon)$ and $Q(\lambda - \epsilon)$. In the event that $Q(\lambda + \epsilon)$ is larger, we keep doubling the value of $\epsilon$ in the increasing direction (i.e. computing $Q(\lambda + 2 \cdot \epsilon)$, $Q(\lambda + 4 \cdot \epsilon)$, etc.), until the process of doubling no longer leads to a further increase in the objective function value. Let $\lambda' = \lambda + r \cdot \epsilon$ be the second-last value of $\lambda$ achieved with the use of this method. Then, we perform a traditional binary search in the range $[\lambda_0, \lambda']$ in order to determine the optimal value of the parameter $\lambda$. A same argument applies in the event that we perform the doubling with reducing value of $\lambda$. As in the previous case, we use the doubling process to determine the exact range $[\lambda', \lambda_0]$ in which the true value of $\lambda$ lies. We perform binary search again in order to determine the optimal value of $\lambda = \lambda^*$. We note that the binary search in the successively smaller intervals is terminated, once the size of the narrowed interval is less than $\epsilon$. Although it is possible in theory for $\lambda$ to be negative, it often does not occur because the link structure is typically positively correlated to label correlations. The overall procedure is illustrated in Figure 1.

### A. Utilizing Factorized Matrices for Classification

We note that the process of matrix factorization creates a set of clusters which can be directly used for classification. Let us consider an unlabeled edge, which is associated with document vector $\overline{d'}$ and link vector $\overline{e'}$.

If the unlabeled edge is included in the original training data set, the cluster membership of this unlabeled edge is already known and classification is a simple matter. The highest frequency class of this cluster is reported as the relevant class label. On the other hand, if the unlabeled edge is not included in the original training data set. This can happen in dynamic scenarios in which new edges arrive into the network at a later stage. This is because the edges may represent communications between nodes which occur continuously over time. For the unlabeled edge, we use the same optimal value of $\lambda = \lambda^*$ (as used in the matrix factorization process) in order to create a content and link (row) vector representation $\overline{s'} = [d', \lambda^* \cdot e']$. Note that the latent characteristic vectors of the different clusters are defined by the rows $\overline{r_1} \dots \overline{r_q}$ of $V(\lambda^*)$. Therefore, we determine the projection of $\overline{s'}$ upon each of these latent characteristic vectors in order to determine which cluster best fits the vector $\overline{s'}$. In order to do so, we compute the dot products $\overline{r_1} \cdot \overline{s'} \dots \overline{r_1} \cdot s'$. The index of the largest dot product provides the relevant cluster for the edge vector $s'$. The dominant class label in this cluster is reported as the relevant one for the edge $e$.

## III. Local Partitioning Speedups

The matrix factorization approach becomes increasingly slow for graphs of larger size. One possibility to speed up the approach is to use smaller *structural localities*. Thus, in effect, this process creates *two levels of clustering*, where the higher level is a very fast randomized partitioning, which is biased towards partitioning the graph into dense structural localities, whereas the lower level is a more fine grained clustering using both structure and content with the use of matrix factorization. In order to distinguish between the higher level of local partitioning, and lower level of matrix-factorization based clustering, we will use the terms *partitions* and *clusters* respectively throughout this section, and the experimental sections. The word *partitions* refers to the fast randomized approach on the larger data set, whereas the term *clusters* refers to the more fine grained approach with matrix factorization.

For the higher level partitioning, we partition the graph into disjoint sets of nodes, and including all edges incident upon these disjoint sets in the corresponding matrices. We note that some of the edges may be incident upon two different partitions, when both of the end points are not fully included in a partition. In such cases, the edge belongs to both partitions. The idea here is to create *tightly connected structural regions* which retain structural information about the underlying edges. The edge membership to clusters is essentially induced by the underlying node membership. Thus, the first step of the problem is to create a partitioning of the node set $N$ into the sets $N_1$, $N_2$, ... $N_r$ which that each node set is a tightly connected region of densely populated edges.

Each of the node set $N_i$ *induces* an edge set $A_i$. The edge set $A_i$ is essentially the set of all edges incident upon the nodes in $N_i$. We note that the sets $A_1 \dots A_r$ may not necessarily be disjoint, because some of edges may have their endpoints in different partitions. Furthermore, some of the nodes not originally contained in $N_i$ may be included in the end points of the induced edge set $A_i$ because of the edges which span multiple partitions. Such overlapping edges may often be relatively limited, especially since most of the edges lie within the dense structural partitions. One characteristic of the sets $A_1 \dots A_r$ is that they retain structural locality much better that a sample based partitioning. Before describing the partitioning process, we will explain how it is used in classification.

As in the previous case, we perform the matrix factorization separately for each partition of edges $A_i$ and create a corresponding set of clusters. As in the previous case, we need to create a matrix $S_i(\lambda_i)$ for each partition, which includes both the effects of structure and content. We determine the corresponding locally optimized value for $\lambda_i$. Unlike the sampling approach in which all partitions are used for potential classification, we need to use only one or two partitions in this case, corresponding to the partition(s) on which the edge is incident. For the case when the edge belongs to only one partition, we determine the majority label in the cluster to which the edge belongs. This label is reported as the class label of the edge. In the event that edge belongs to two partitions, we report the majority label of the union of the two clusters to which the edge belongs.

It now remains to explain how the node set is partitioned into the sets $N_1 \dots N_r$. The best partition is one which is *reasonably balanced* in terms of the number of edges incident on them. This ensures that the sets $A_1 \dots A_r$ are approximately of the same size, and therefore the matrix factorization process of any particular set is not a bottleneck on the efficiency of the method. In order to keep track of the edge set size associated with each partition, we observe that the number of edges incident on a partition is (approximately) equal to half the sum of the degrees of the nodes in the partition. This is an approximation because the two end points of some of the edges may lie in different partitions. Therefore, we use the degrees on the nodes as weights, and use these weights in order to generate approximately balanced partitions.

We use an edge-sampling approach for the node-partitioning process. The edge-sampling approach essentially samples edges from the graph and constructs clusters which are the connected components of this sample of edges. This edge sampling approach is biased towards finding densely connected regions of edges [3] because the edges are more likely to be sampled from the dense regions of the graph. It has been shown in [3] that the use of edge sampling can be used to approximate a minimum 2-cut with high probability if repeated samples are used, and the best of these repeated samples is selected. As in [3], we can also use the repeated samples in order to further improve the quality of the clusters.

The main challenge here is to modify the sampling approach, so that the node partitions are weighted approximately evenly. The first step is to generate subsets of the $r$ different clusters. Since we want to partition into $r$ different clusters, we do not allow the weight of any connected component to be more than $1/r$ of the total weight of the edges. We start off with isolated nodes of the network and add edges one by one in random order to order to grow the underlying components. The only exception is that an edge is *not* added, when its addition would cause the weight of the resulting component to be more than $1/r$ of the total.

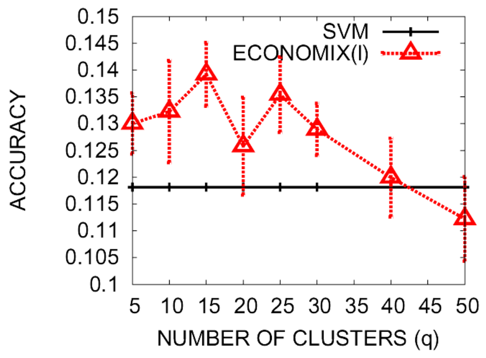
At the end of the first scan of the edges, we have a set of connected components, the size of each of which is at most $1/r$ of the total network size. At this point, we pick the $r$ largest components of the network, which serve as the starting points to completely grow the underlying clusters. We refer to these components as primary, and the remaining smaller components as secondary. All secondary node sets which are not one of these $r$ largest components are then assigned to one of these $r$ primary components. In order to decide the primary component to which a secondary component may be assigned, we compute the number of edges which connect the secondary component to each of the primary components. The secondary component is then assigned to the primary component to which it is connected with the most number of edges. In the event that the secondary component is not connected to any of the primary components with even one edge, we skip that component for the time being, and proceed with the next secondary component.

This process is performed until all secondary components have been processed at least once. Then, we perform one more pass on the secondary components with the same approach and try to connect them to one of the primary components. For a connected (base) graph, each such pass will attach at least one or more secondary components to a primary component. This approach is repeated until all secondary components have been assigned. The resulting node partitions are $N_1 \dots N_r$. These are used to generate the edge partitions $A_1 \dots A_r$. The quality of the node sets $N_1 \dots N_r$ is better when the end points of few edges lie in different partitions. In order to improve the quality, we performed the entire sampling process multiple times (5 to 10 times in our experiments), and selected the partitioning which results in the least number of cross-partition edges.

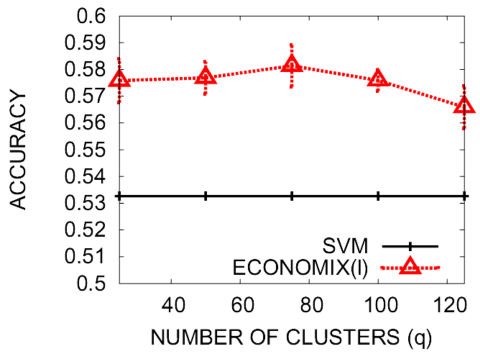
## IV. Experimental Results

Because of lack of space, we provide a brief overview of the the key results. We used an *Enron* data set[1] in which 18 keywords in subject lines of emails were used to generate the edge classes. We also used a *Twitter*[2] data set with 100,000 nodes from year 2010 in which 14 hash tags were used to generate class labels on the edges. The SVM classifier[3] was used as a baseline with the bag-of-words representation.

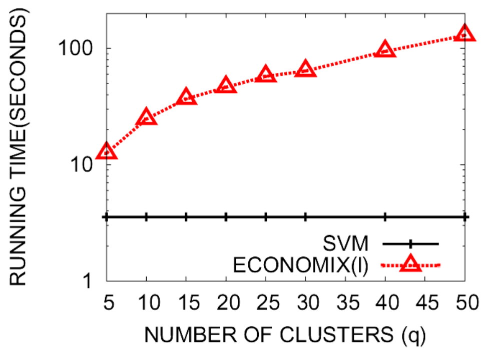
The results of *ECONOMIX* for the *Enron* and *Twitter* data sets are shown in Figures 2(a) and (b), respectively. Note that the number of clusters refers to the factorization rank, whereas the number of partitions refers to the sampling-based partitioning [3]. The *Twitter* data set contained a larger number of nodes and therefore it required a greater number of partitions. Furthermore, too many clusters/partitions lead to overfitting in *Enron* because of the small data size. In both cases, *ECONOMIX* performed better than the baseline SVM classifier. The efficiency results are shown in Figure 2(c). Although the SVM approach is faster (because it uses only content), the *ECONOMIX* approach was still reasonably efficient.



(a) *Enron* accuracy (10 partitions)



(b) *Twitter* accuracy (50 partitions)



(c) *Enron* efficiency (10 partitions)

Fig. 2. Accuracy and efficiency of *ECONOMIX*

## References

[1] C. Aggarwal, G. He, and P. Zhao. Edge classification in networks, *ICDE Conference*, 2016.

[2] J. Tang, Y. Chang, C. Aggarwal, and H. Liu. A survey of signed network mining in social media. *ACM Computing Surveys*, 49(3), 42, 2016.

[3] A. A. Tsay, W. S. Lovejoy, and D. R. Karger. Random Sampling in Cut, Flow, and Network Design Problems. *Mathematics of Operations Research*, 24(2), pp. 383–413, 1999.

[1]http://www.cs.cmu.edu/~enron/
[2]http://www.twitter.com
[3]http://weka.sourceforge.net/doc/weka/classifiers/functions/SMO.html