Inductive representation learning on feature rich complex networks for churn prediction in telco

María Óskarsdóttir[1], Sander Cornette[2], Floris Deseure[2], and Bart Baesens[2,3]

[1] Dept. of Computer Science, Reykjavík University,
Menntavegi 1, 101, Reykjavík, Iceland,
mariaoskars@ru.is,
[2] Dept. of Decision Sciences and Information Management, KU Leuven,
Naamsestraat 69, 3000 Leuven, Belgium,
[3] Dept. of Decision Analytics and Risk, University of Southampton, United Kingdom

Abstract. In the mobile telecommunication industry, call networks have been used with great success to predict customer churn. These social networks are complex and rich in features, because the telecommunications operators have a lot of information about their customers. In this paper we leverage a novel framework called GraphSAGE for inductive representation learning on networks with the goal of predicting customer churn. The technique has an advantage over previously proposed representation learning techniques because it leverages node features in the learning process. It also features a supervised learning process, which can be used to predict churn directly, as well as an unsupervised variant which produces an embedding. We study the impact that including node features has when predicting churn as well as the benefit of a complete learning process, compared to an embedding with supervised machine learning techniques. Finally, compare the performance of GraphSAGE to that of standard local models.

Keywords: Call network, churn prediction, representation learning, supervised learning

1 Introduction

The mobile telecommunication (telco) industry is a competitive market as cell phone ownership has reached saturation in the developed world and customers can easily change telco providers. Because attracting new customers is more expensive than retaining current ones, telcos focus their customer relationship management on keeping their existing customers happy and making offers to customers whom they think are at a risk of leaving —or churning— in order to persuade them to stay. These potential churners are usually identified by means of churn prediction models which are typically binary classifiers based on input features provided by the telco . The goal of the classification is to identify which customers are most likely to churn.

In recent years, social influence has been shown to affect churn, in the sense that if someone churns they are likely to persuade others to churn as well [5]. There are various ways to predict churn with social networks. One can extract network features from the social network and use them in a traditional binary classifier, such as logistic regression. These features describe the neighborhood of a customer and their position in the network using various numerical attributes. The features are added to the customer dataset and can positively influence the performance of a churn prediction model. Relational learners, on the other hand, simulate the spread of churn influence through the network and thus learn churn probabilities for customers directly from the network [5],[9]. Research shows that combining relational learners with network and local features gives the optimal performance [7]. Because the type of churners detected by non-relational and relational classifiers are different, more churners can be detected by combining both classifiers in a single combined model. Although these approaches are capable of incorporating social network effects, yielding higher predictive performance, they do require sophisticated feature engineering in order to capture the desired effects. However, this process can be time consuming, unfeasible on large networks and the resulting features not capable of encapsulating the social networks effects completely.

An alternative approach, that overcomes this downside, is representation learning, where the idea is to encode the nodes' characteristics and their position in the network in a low dimensional vector space. Instead of trying to encode network information through feature engineering, such techniques learn this information directly from the network. The main idea behind these state-of-the-art methods is that similar nodes should have a similar representation. In recent years some new scalable ways to transform data into network features have arisen. DeepWalk, for example, applies the skip-gram approach of word2vec to random walks on networks [8]. The idea of word2vec is to gather information of a word by representing the it with a vector of strongly related words. Moreover, in node2vec, customizable two-step random walks allow for an intelligent sampling of the nodes' neighborhoods before generating the embeddings [2]. Recently, node2vec was applied for customer churn prediction in telco and adapted to the dynamic aspect of churn [6]. These methods are called shallow embedding approaches, meaning that a big matrix is created, containing all the embedding vectors of all the nodes [4]. The encoding function can be compared to a simple lookup function. This approach has a couple of downsides. Firstly, no trainable parameters are shared which can be inefficient, both statistically and computationally. Secondly, the methods do not incorporate local node features, while these might contain valuable information. Finally, shallow embedding approaches such as DeepWalk and node2vec are inherently transductive which mean that they are not able to generalize over unseen nodes.

GraphSAGE, however, is a novel framework for inductive representation learning that is capable of generating node representations based on node attribute information [4]. In this paper, we apply GraphSAGE to the call networks of telco customers with the goal of predicting churn. GraphSAGE has both an unsupervised and a supervised variant. The unsupervised variant learns a node embedding that can be supplied to a downstream machine learning algoritm, such as decision trees or k-means clustering. GraphSAGE can also be trained in a fully supervised way, thus producing a prediction for each node. GraphSAGE is novel in the the sense that it incorporates node features when learning the embedding functions. It therefore works particularly well for feature rich networks but it can also be applied to networks without any features. In this paper we demonstrate that including even just a handful of features can improve the performance greatly. We explore the efficiency o the two variants of GraphSAGE, supervised and unsupervised, in this regard, as well and investigating how many node features are needed in addition to the network structure, to obtain good predictive performance.

The rest of this paper is organized as follows. In the next section we explain the GraphSAGE framework. In Sect. 3 we describe the setup of our experiments and report the results in Sect. 4. Finally, we conclude the paper in Sect. 5.

2 GraphSAGE

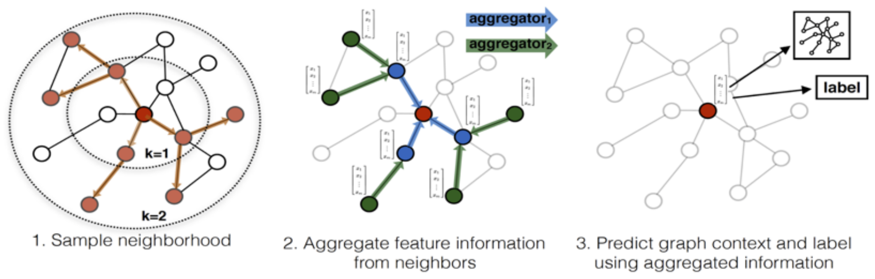
aggregator₁
aggregator₂
k=1
k=2
label
1. Sample neighborhood
2. Aggregate feature information from neighbors
3. Predict graph context and label using aggregated information

Fig. 1. The GraphSAGE framework. Image source [4].

The GraphSAGE framework was developed as an inductive alternative to DeepWalk and node2vec that is furthermore able to learn functions of node embeddings and hence generalize across different networks and unseen nodes [4]. It simultaneously learns the topological structure of each node's neighborhood and the distribution of node features in the neighborhood. It trains a set of aggregator functions that learn to collect and combine feature information from a node's neighborhood. These aggregator functions can then be applied to unseen data.

The GraphSAGE framework, seen in Fig. 1, consists of two parts. Firstly, the forward propagation or embedding generation algorithm. This algorithm generates embeddings for nodes assuming that the different GraphSAGE model parameters have already been learned. Secondly, the learning of the GraphSAGE model parameters. For the embedding generation, two sets of parameters are assumed to be learned. The first set of parameters are those from the K aggregator functions that aggregate the information of a node's neighborhood. Theoretically, the neighborhood of a node could be all the nodes which are connected with an edge to that node. However, this means that the neighborhood of a node can be very large and could increase the computation time substantially. Therefore, GraphSAGE uniformly samples a fixed amount of neighbors for each node by using random walks through the networks. The second set of parameters are a set of weight matrices, $W^k, \forall k \in \{1, \ldots, K\}$. These weight matrices are used to propagate information between the different search depths, or to put it more plainly, to propagate node information to the preceding node in the random walk. At the start of the embedding generation algorithm, the representation of each node is the features that are provided as input. At each depth k in the search depth ($k \in \{1, \ldots, K\}$) and for each node v we aggregate and concatenate the information from the neighborhood of that node v. At the aggregation step, the neighborhood of each node is aggregated by an aggregation function, represented as a vector $\mathbf{h}^k_{\mathcal{N}(v)}$. However, it should be noted that the representations of the nodes in the neighborhood of node v depend on the representations generated in the previous search depth $(k-1)$. After this aggregation, the representation of the node's neighborhood $\mathbf{h}^k_{\mathcal{N}(v)}$ and the node's current representation $\mathbf{h}^{k-1}_{(v)}$ are concatenated. Using σ as a nonlinear activation function and passed through a fully connected layer. This results in the representations of each node $\mathbf{h}^{k-1}_{(v)}$ which can be used in the next step of the iteration. Lastly, at depth K the node representation $\mathbf{h}^K_{(v)}$ is denoted as z_v for notational convenience. The GraphSAGE algorithm can also run without any local features where instead it uses only structural features, such as the node's degree.

The learning of the parameters is different for the unsupervised and supervised variant. For the unsupervised variant, a graph-based loss function is applied to the node representation z_v. This loss function ensures that nearby nodes have a similar representation and nodes further away are distinct. In the supervised variant of GraphSAGE, the unsupervised loss function is replaced with the supervised cross-entropy loss function, although other supervised loss functions could also be implemented. The difference between these loss functions is the main difference between the unsupervised and supervised variant of GraphSAGE. The unsupervised loss function returns representations, which can be used with other classifiers and are useful for multiple prediction settings. The supervised loss function, on the other hand, only works for one specific classification setting, namely the target that is provided for the learning. Nevertheless, for both the unsupervised and supervised setting the parameters of the aggregators and the weight matrices are tuned by using stochastic gradient descent.

There are three types of aggregator functions in the GraphSAGE algorithm. The first type is the mean aggregator. Two different aggregators are included for this type. The first aggregator takes the elementwise mean of the different features for each node. The second aggregator is the graph convolutional network or GCN aggregator. This aggregator is closely related to the first one, but is a convolutional aggregator. This is also the only aggregator that does not have concatenation step despite the fact that this step leads to significant performance gains The second type of aggregator is the long short-term memory or LSTM aggregator. This aggregator is based on an LSTM architecture. Compared to the mean aggregator this aggregator is more complex and has more expressive power. The third type of aggregators are the pooling aggregators. For this aggregator the representation vectors of the neighbors of a node are independently passed through a fully-connected neural network which transforms these representations, after which an elementwise pooling operation is done for the whole neighborhood in order to aggregate the information. The two different pooling operators included in GraphSAGE are max- and mean-pooling. Where max-pooling takes the elementwise maximum, the mean-pooling operation takes the elementwise mean. Ideally, all these aggregators should be symmetric. The ordering of the neighbors of a node should not matter because these are by default unordered. Aggregators should also be trainable and have high representation power. However, only the pooling aggregators are both symmetric and trainable, the mean aggregators are obviously not trainable because they take the elementwise mean and the LSTM aggregator is not symmetric because it sequentially processes its inputs.

The advantage of using GraphSAGE is that it solves problems that arise when working with large networks and that it is an inductive approach. Because the number of parameters is fixed, the process is scalable. The model is also independent of the network structure. Once the parameters are learned, the model can be applied on different dynamic or even unseen networks by using the learned aggregator functions.

3 Experimental Setup

3.1 Preprocessing the data

For our study we used call detail records (CDR) of over a million customers with a prepaid contract provided by a Belgian telco. The data spans a total of six months and describes the calls made and received by the customers. In addition we have local customer features that provide more information about the customers present in the CDR data. The local features are aggregated at a monthly level.

The first step in applying GraphSAGE on a social network, is constructing the networks themselves. A social network is created from nodes and edges. In our case each node represents a customer of the telco and each edge represents a phone call between two customers. We consider undirected networks, meaning that the relationship between two customers goes both ways instead of only from the calling customer to the receiving customer. In line with the literature, we disregard calls that last less than four seconds. For our experiments, we created a separate network for each month from the CDR data. For each customer we also include the local features and the label for each month. A description of each network can be found in Table 1.

Table 1. Network description

Month	Number of nodes	Number of edges	Churn rate
M1	772 895	4 590 812	17.44%
M2	771 690	4 501 831	20.62%
M3	738 639	4 140 595	15.73%
M4	736 309	4 015 522	16.69%

We use a binary label that indicates whether or not the customer churns in the following month. For customers with prepaid contracts, churn is often passive, i.e., the customers do not cancel their subscription but instead simply stop using their numbers, and as a result the date of churn is not known exactly. We adopt a churn definition that is common in the literature where a customer is labeled as a churner if they do not make or receive a phone call for 30 consecutive days, and then churn date is then the first day in which they are inactive [7]. This means, that to generate the churn label for the first month, we need the data from the next two months. In order for a customer to be labelled as a churner, 30 days of inactivity are required. Since the churn date is in M2, we need to check in M3 whether the 30-day inactivity period is met. As a result, we are able to generate four networks from six months of data. The first two will be used as training data, the third as validation set and the fourth as test set.

We create various sets of feature to test the effect node features have on the predictive performance. For both the unsupervised and the supervised variant of GraphSAGE, we consider 3, 5, 7, 10, 15, 30 ,50 ,70 and 90 features. The feature selection method used is the recursive feature elimination or RFE [3]. It selects features by recursively eliminating the least important feature using Logistic Regression and eliminating that feature from the entire set of features until only the desired amount of features is left. At the end of this process we have the rank of each feature which can then be selected to be used in the GraphSAGE algorithm depending on the amount of features that should be

3.2 GraphSAGE setup

In the GraphSAGE framework, there are various hyper-parameters that need to be set. First are the number of random walks and the depth, which have default values 50 and 5, respectively. However, to avoid excessively long runtimes we use 25 walks with depth 2, as higher values only result in marginal performance gains [4]. For the mini batches of the random walks we use the default setting of sample sizes $S1 = 25$ and $S2 = 10$. Both unsupervised and supervised Graph-SAGE use an aggregator function to concatenate the information from the local neighborhood of each node and generate the node embeddings. As part of our experiment we test all the aggregator function but only report the best result in each case.

The supervised GraphSAGE learns churn probabilities directly from the networks using a task specific cross-entropy loss function. In contrast, the unsupervised GraphSAGE applies a graph-based loss function that ensures that nearby nodes have a similar representation that is distinct from the representation of nodes that are further away. The representation can then be used by supervised learning techniques to predict churn. We use logistic regression, random forests and neural networks, as they provide alternative view on the trade-off between model complexity and predictability and are widely used in the churn prediction literature and industry [7]. We tune the hyper-parameters of these techniques using the validation set. Instead of using a graph-based loss function, the supervised GraphSAGE uses a cross-entropy loss function. Contrary to the unsupervised variant, the supervised variant does not learn representations but it immediately classifies the nodes by using this task specific

3.3 Benchmark models

To create a benchmark for the results we obtain with GraphSAGE, we build models with local features only. More specifically, we use the same classifiers as those used in the unsupervised version of GraphSAGE. These are logistic regression, random forests and a neural networks. These models are built in the same manner as we built the GraphSAGE models, whereby M1 and M2 are two independent networks and are used as training data. The third month is used for validation purposes and the fourth month for testing purposes. However, the local features are loaded in directly. Instead of generating these feature vectors, which also incorporate social network effects, with GraphSAGE. In order to be able to make a fair comparison we tune the hyper-parameters for these models using grid search on the validation set.

3.4 Performance measurement

We use three commonly used performance measures to evaluate the performance of our churn prediction models [7]. The lift measure represents how much better a model is at identifying churners than one would find in a random sample. We choose lift at 0.5% and 5% to obtain a slightly alternative view on the performance. The first lift measure considers only a very small fraction of the customers who have the highest predicted probability of churn giving a more realistic view for large customer bases. In contrast, the latter lift measure considers a larger sample and gives a more holistic view on the performance. We also consider the well known area under the receiver operating characteristics curve (AUC) which represents in a single number the trade off between specificity and sensitivity of the model.

4 Results

4.1 Comparison of feature performance

First we study the effect of including features when learning the embedding function. Figure 2 shows the performance of the supervised GraphSAGE (yellow)

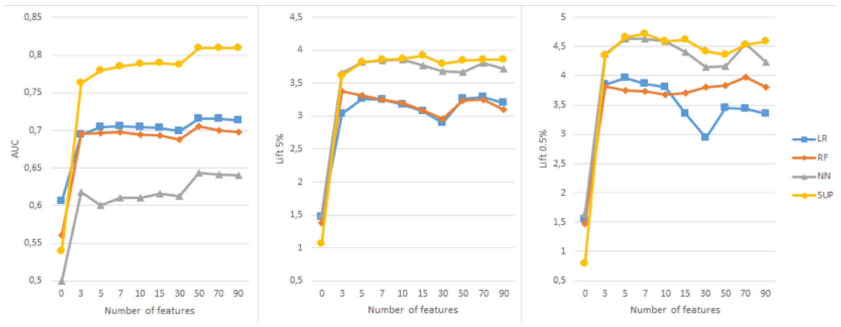

Fig. 2. Impact of features on the performance of GraphSAGE.

as well as the unsupervised GraphSAGE together with logistic regression(blue), random forests(orange) and neural networks(gray) as a function of number of features. The performance is measure in AUC, lift at 5% and lift at 0.5% These results were obtained using embeddings generated by GraphSAGE mean-based aggregator. For these results the unseen test network was used for the first time.

Noticeably, the performance of GraphSAGE without any features is much lower than with even a single feature. For example, the AUC of the supervised GraphSAGE goes from 0.538 to 0.764 when adding three node features. Similarly, the 0.5% lift of the unsupervised GraphSAGE with random forest is 1.473 with no node features and 3.817 with three. After this initial significant jump in performance when adding features, the performance keeps increasing slightly when adding more features, but decreases again in some cases. This shows that more features are not always better, but a certain number of good features help the churn prediction significantly. Additionally, including more features increases the computation times so it is important to find a good balance between number of features and the associated performance gain. Based on these results, we can estimate that including five to fifteen features is sufficient.

4.2 Comparison of model performance

Table 2. The performance of GraphSAGE models and local models.

Model	Variant	AUC	5% lift	0.5% lift
GraphSAGE	Unsupervised	0.614	3.857	4.644
	Supervised	0.794	3.892	4.797
	Neural Network	0.5	3.128	3.902
Local	Random Forest	0.709	3.796	4.407
	Logistic Regression	0.653	1.582	3.843

We also look at the performance of the two variants of GraphSAGE with respect to each other and in comparison to local models. Table 2 shows the performance of both GraphSAGE models and the performance of three models, that is, neural networks, random forests and logistic regression trained with local features. The performance is measured in AUC and lift at 5% and 0.5%. Evidently, the supervised GraphSAGE has the highest predictive performance according to all evaluation measures. The second best model is the unsupervised graphSAGE when measured by the two lift measures. The best local model is the random forest.

The fundamental difference between the local and the unsupervised GraphSAGE models is that the features in the first case are local, that is, specific for each customer without taking into account the feature values of neighbors. In contrast, the embedding that the unsupervised GraphSAGE learns, is rich in information about the properties of neighbors, which evidently is important in this context for homophily and social influence because of the increase in performance.

5 Conclusion

GraphSAGE is a novel framework for inductive representation learning on networks that is furthermore capable of taking into account node features when learning node embeddings. In this paper, we applied GraphSAGE to a feature rich call network to predict customer churn in telco. It is evident from our results that the node features add valuable information when learning node embeddings as seen by the great leap in performance between models with and without local features. We also saw that only a few features are needed for a high improvement in performance. Furthermore, the supervised variant of GraphSAGE which learns churn probabilities directly achieved the best performance. The unsupervised variant performed better than the local models when measured in both lift measures. This indicates that the embedding learnt by GraphSAGE possesses more valuable information for this prediction task than the local features alone. That is, the feature values of neighboring nodes are important. This furthermore establishes the importance of social influence and homophily in churn prediction. We have shown that GraphSAGE is a feasible approach for churn prediction with call network.

In future work, we will apply the algorithm to call networks with a different target variable, such as customer lifetime value. In that way we could leverage the inductive nature of GraphSAGE to its full extent, and extract node embeddings to unseen nodes. We would also like to extend the framework for dynamic networks.

References

1. Backiel A, Verbinnen Y, Baesens B, Claeskens G. Combining local and social network classifiers to improve churn prediction. In: Proceedings of the 2015 IEEE/ACM

international conference on Advances in Social Networks Analysis and Mining 2015, pp. 651-658. ACM (2015).
2. Grover A, Leskovec J. node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining 2016 , pp. 855-864. ACM (2016).
3. Guyon I, Weston J, Barnhill S, Vapnik V. Gene selection for cancer classification using support vector machines. Machine learning. 2002 Jan 1;46(1-3):389-422.
4. Hamilton W, Ying Z, Leskovec J. Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems 2017, pp. 1024-1034 (2017).
5. Kim K, Jun CH, Lee J. Improved churn prediction in telecommunication industry by analyzing a large network. Expert Systems with Applications. vol. 41, no. 15, pp. 6575-6584 (2014).
6. Mitrovic S, Baesens B, Lemahieu W, De Weerdt J. tcc2vec: RFM-informed representation learning on call graphs for churn prediction. Information Sciences. (2019).
7. Óskarsdóttir M, Bravo C, Verbeke W, Sarraute C, Baesens B, Vanthienen J. Social network analytics for churn prediction in telco: Model building, evaluation and network architecture. Expert Systems with Applications. vol. 85, pp. 204-20 (2017).
8. Perozzi B, Al-Rfou R, Skiena S. Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining 2014, pp. 701-710. ACM (2014).
9. Verbeke W, Martens D, Baesens B. Social network analysis for customer churn prediction. Applied Soft Computing. vol. 14, pp. 431-46 (2014).