CHURN DETECTION IN LARGE USER NETWORKS

Joya A. Deri and José M. F. Moura

Carnegie Mellon University
Department of Electrical and Computer Engineering
Pittsburgh, PA 15213 USA
E-mail: {jderi,moura}@ece.cmu.edu

ABSTRACT

Anomaly detection on dynamic real-world networks such as large caller networks and online social networks is a very difficult problem, analogous to looking for a needle in a haystack. This paper considers detecting churners in a 3.7 million mobile phone network. The two main issues are designing fast and efficient features and classifiers. We discuss both in this paper. We associate every caller in the network with an activity vector and an affinity graph, and our features are derived from activity levels computed from subgraphs of the affinity graph. These features reflect the graph-dependent nature of the problem. To compute these networks expeditiously, we extend as integral affinity graphs the concept of integral images. Our anomaly classifier is a cascaded classifier with stages that combine naive Bayes and decision tree classifiers. Simulations with a 3.7 million cell phone user network illustrate an anomaly classifier that reaches a false alarm rate of 0.8% with a churn detection rate of 71%.

***Index Terms*—** anomaly detection, large-scale dynamic networks, integral image, integral affinity graphs, cascaded classification

1. INTRODUCTION

Anomaly detection on large user networks is complicated not only by the computational issues imposed by the size of the data, but also by similarities between anomalous and non-anomalous behavior. For mobile service providers with millions of subscribers, isolating the "churners" (the small percentage of customers who will drop their carriers) is a challenging problem, especially for customers that do not have a fixed period service contract that commits them to the service provider. Carriers would like to identify potential churners before they actually churn; in this way, they can better target advertising and design incentives to prevent or compensate for decreases in their consumer base.

In this paper, we are interested in detecting churners *before the fact* in a network of 3.7 million prepaid cell phone users. Our goal is to design a classifier that can flag these potential churners. In the literature, churn detection has been approached via signal processing techniques such as Kalman filtering [1] as well as machine learning [2], for example. In this paper, we seek to use the network structure of the data to train and test classifiers to identify churners. Anomaly detection for networks has been studied in [3, 4], both of which use neighborhoods of vertices to detect anomalies. In [3], neighborhoods in bipartite graphs are explored via random walk-type methods to identify vertices that participate in multiple non-overlapping neighborhoods. In [4], weighted adjacency matrices for localized subgraphs are used to compute outlier statistics corresponding to anomalies in networks with up to 1.6 million vertices. Anomaly detection for identifying faces in images with a cascaded classifier is discussed in [5].

We address our problem of churn detection as follows. We first develop a graph representation of the dataset where vertices represent callers and edges connect callers who call each other. This graph structure is one of the key ingredients of our method. Secondly, we develop a set of fast and efficient features. For a network vertex A, we start by considering a 16-dimensional activity row vector that collects several usage statistics between a caller and its neighbors. Then, we construct samples of size M for each of the neighbors of A and for each of the neighbors of neighbors of A via a snowball method (see Section 3). These M callers form a subgraph that we refer to as the *affinity graph* of A, with an associated $M \times 16$ activity matrix consisting of the activity vectors of its vertices. We have tested our classifiers with $M = 20$, 30, 40, 50, and 100. Next, we construct an expanded set of features considering subgraphs of the affinity graph, and we compute activity vector sums for the callers within each subgraph to get the *subgraph activity*. We use the differences between subgraph activities as the features to classify vertex A.

To expedite feature computation, we adapt the concept of integral images from [5] to allow quick computation of subgraph activities and their differences. We use these features to build a cascaded classifier to decrease the false alarm rate.

The rest of the paper is as follows. We present the network construction in Section 2. In Section 3, we define our feature set and extend integral images to arbitrary networks.

This work has been supported by AFOSR grant FA95501010291 and by NSF grants CCF1011903 and CCF1018509.

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

Sections 4 and 5 discuss a cascaded classifier implementation with experimental results. We conclude in Section 6.

2. NETWORK CONSTRUCTION

We have available 11 consecutive months of cell phone activity for a caller network with over 3.5 million callers in each month. In this paper, we consider only the data for the months January and February 2009, each with 3.7 million callers. We construct networks for these two months of data and use the first month for training and the second month for testing the anomaly classifier. The networks for January and February are denoted by $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ respectively, where V_t is the set of vertices and E_t is the set of undirected edges, $t = 1, 2$. The vertex set V_t includes all callers of the network at month t that make at least one in-network call in month t. An edge between two callers in V_t exists in E_t if there is at least one call between them in month t.

3. SUBGRAPH-ACTIVITY BASED FEATURES

We first describe the activity vectors associated with each caller in our network and the subgraph-activity features we compute. Then we extend the concept of integral images from [5] to graphs. We describe how we extend this concept first to trees and then to networks of arbitrary structure. We call the result the *integral affinity graph*.

Activity Vectors and Features. We associate four classes of activity to every caller $A \in V_t$: calls initiated by A to an in-network user $B \in V_t$; calls received by A from an in-network user $B \in V_t$; calls to an out-of-network user $C \notin V_t$; and, lastly, calls received from an out-of-network user $C \notin V_t$. For each of these possibilities, we account for the corresponding number of calls, total call time, total number of SMS messages, and the number of callers. The resulting activity vector of dimension 16 is recorded for every $A \in V_t$.

In addition, each caller $A \in V_t$ has an associated *affinity graph*, denoted by $G_A = (V_A, E_A)$, $V_A \subset V_t$, $E_A \subset E_t$. The affinity graph for caller A is constructed by performing a snowball sample [6]. First, A is added to vertex set V_A. Then, we perform snowball sampling to collect its neighbors, which we call the first-wave neighbors. If the target sample size M has not yet been reached, we snowball sample again to collect the neighbors of the first-wave neighbors, which form the second-wave neighbors. We continue in this manner until M vertices have been chosen [6]. We collect the activity vectors in V_A to form an $M \times 16$ activity matrix U_A associated with caller A. As discussed in [7, 8, 9, 10, 11], sampled networks can preserve and discover global network properties such as degree distributions. In addition, [11] shows that even biased estimates derived from network samples without reweighting can reflect global properties. While we are only interested in preserving the local structure of a caller A when we construct its affinity graph, the idea that subsets of the affinity graph reflect global properties motivates our decision to examine subregions of the affinity graph G_A to build our feature set. In particular, our features of interest are the differences of the activity vectors of adjacent subgraphs of G_A.

To compute the features, we first define the following: Let r, $r : V \to \mathbb{R}^{16}$, represent the activity vector sum, and let its ith entry be the sum of the ith entries of the activity vectors of the vertices in a subgraph $G_A = (V_A, E_A)$, i.e.,

$$r_i(G_A) = \sum_{v \in V_A} U_A(v, i), \tag{1}$$

where $U_A(v, i)$ denotes the ith entry of the row for caller v in the activity matrix U_A. Suppose $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are two connected subgraphs of G_A. We restrict G_2 to the subgraphs of G_1. For pairs of such subgraphs, our set of features F_A for an affinity graph G_A is the set of activity sum differences $F_A = \{r(G_{A,1}) - r(G_{A,2})\}$ for the subgraphs. In Figure 1(b), one example subset of features would be $\{r(T_1) - r(T_i) \mid i \in [2, 10]\}$. To efficiently compute these features for affinity graphs, we employ the notion of integral images from [5] and extend it to general graphs.

Background on Integral Images. Consider an image $G = (V, E)$ – i.e., G is a finite two-dimensional lattice. Vertices are labeled in lexicographic order, starting from the top left corner vertex as vertex 1 and proceeding sequentially from left to right and top to bottom. With this lexicographic order, the integral image $z(v)$ replaces the image pixel value $U(v)$ at pixel v with the pixel sum of all vertices above and to the left of v, i.e.,

$$z(v) = \begin{cases} U(v) & \text{if } v = 1, \\ U(v) + \sum_{v' < v} z(v) & \text{otherwise.} \end{cases} \tag{2}$$

For example, the pixel sum of window D in Figure 1(a) can be computed in four array references: $z(1)+z(4)-(z(2)+z(3))$.

Integral Affinity Trees. We extend the integral image concept to trees. We first provide the following standard definitions for a graph $G = (V, E)$ [12]. A *path* exists between any two vertices $v, w \in V$ if a sequence of edges in E connects them. A *(directed) tree* $T = (V, E)$ is a graph such that any two vertices $v, w \in V$ are connected by exactly one path that has no repeated vertices. A vertex $w \in V$ is a *descendant* of a vertex $v \in V$ if there exists a directed path from v to w. We denote by $\mathbb{D}_v$ the set of descendants of v. In addition, the *subtree* T_v of T with root vertex $v \in V$ is the subgraph containing v and descendants $w \in \mathbb{D}_v$.

For an affinity tree graph G_A associated with vertex A, our features of interest are the differences of feature sums in adjacent subtrees, i.e., $\{r(T_v) - r(T_{v'}$ for $v' \in \mathbb{D}_v\}$. To compute these features, we define an *integral affinity graph*:

Definition 1. *Consider an affinity tree graph $T_A = (V_A, E_A)$ for a caller A. Then the* integral affinity graph *z_A at a vertex $v \in V_A$ is defined as the activity vector sum of subtree T_v:*

$$z_A(v) = \sum_{v' \in T_v} U_A(v') = r(T_v), \tag{3}$$

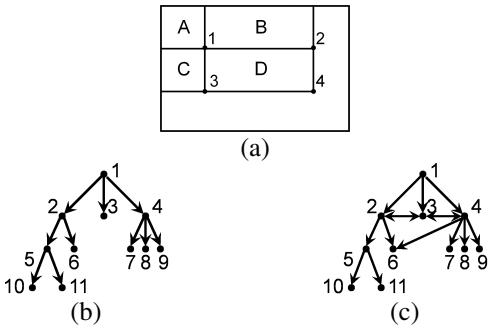

Fig. 1. Example networks to illustrate integral affinity graphs: (a) a 2-D lattice from [5], (b) a tree, and (c) a general network.

where $U_A(v)$ is the activity matrix row corresponding to vertex v and r is the feature sum function in (1).

The integral affinity graph for trees can be expressed by the following recurrence relations:

$$z_A(v) = \begin{cases} U_A(v) & \text{if } |\mathbb{D}_v| = 0 \\ U_A(v) + \sum_{v' \text{ a child of v}} z_A(v') & \text{otherwise} \end{cases}. \quad (4)$$

Computing the difference between two subtree activities is equivalent to computing the integral affinity graph difference at the two roots of the subtrees. In Figure 1(b), for example, the difference between the subgraph activities for subtrees T_2 and T_5 is $z_1(2) - z_1(5)$.

Extension to General Networks. For non-tree affinity graphs $G_A = (V_A, E_A)$, we define the integral affinity graph in terms of a tree-like structure that accounts for inter-level connections. We use breadth first search (BFS) to construct spanning trees, which entails exploring all neighbors of caller A before exploring neighbors of neighbors [12]. We construct the BFS tree T_A with root A and denote the dth *level set* of vertices in G_A with respect to T_A as $L_{T_A,d}$, where $d = 1$ is the root level and d is no more than the maximum depth of T_A. For example, Figure 1(b) shows the BFS tree of Figure 1(c), so the vertices 5 and 6 belong to the level set $L_{T_1,3}$. We account for connections between levels in the BFS tree since we are interested in the distance of subtrees from the root – i.e., the integral affinity graph for vertex 4 in Figure 1(c) will include not only the activity sums for its children, but also for the vertex 6. For simplicity, we do not account for intralevel edges, such as (2,3) or (3,4). If properties such as clustering coefficients are features of interest, they can be included in the activity matrix U_A.

Let Ω_v represent the neighbors of vertex $v \in V_A$ in the arbitrary network G_A. Define a *region* R_v with root vertex $v \in V$ at level i in T_A as

$$R_v = \{v \cup (\Omega_v \cap L_{T_A,i+1})\}. \quad (5)$$

We then have the following definition for general networks:

Definition 2. *Consider an affinity graph $G_A = (V_A, E_A)$ for a caller A. Let T_A denote its corresponding breadth-first-search tree with root A. Then the* integral affinity graph *with respect to T_A for a level-i vertex $w \in V_A$ is defined as the sum of the activity vectors of w and its neighbors in level set $i+1$:*

$$z_A(v) = \sum_{v' \in R_v} U_A(v') = r(R_v), \quad (6)$$

where R_v is the region with root $v \in V_A$, $U_A(v)$ is the activity vector of vertex v, and $r(R_v)$ is the activity vector sum.

For a vertex $v \in L_{T_A,i}$, the corresponding recurrence relations are as follows:

$$z_A(v) = \begin{cases} U_A(v) & \text{if } |\mathbb{D}_v| = 0 \\ U_A(v) + \sum_{v' \in R_v} z_A(v') & \text{otherwise} \end{cases}. \quad (7)$$

The integral affinity graph allows efficient feature computation. Computing the BFS tree has worst-case time complexity $O(|V| + |E|)$ [12], and computing the integral affinity graph requires traversing the BFS tree from the leaves to the root, which has complexity $O(|V|)$. Given the integral affinity graph, we can compute our subgraph activity differences in two array references. For example, the difference between the regions R_1 and R_6 in Figure 1(c) is given by $z_1(1) - z_1(6)$. Since the vertices are in tree levels 1 and 3 respectively, we assign the category *31* to this feature. An affinity graph G_A with a d-level BFS tree T_A will have $\binom{d}{2}$ such feature categories. For 2000 randomly selected vertices from the month 1 network and fixed snowball sample size $M = 50$, we obtain 13,886 *21*-features, 65,407 *32*-features, and 26,692 *42*-features, for example. Note that we can compare networks that have BFS trees with different depths by encoding the lack of a level in the associated feature vector. We use these categories of features to train and test a cascaded classifier for churn detection, which we introduce in Section 4.

4. CLASSIFIERS

We use two types of base classifiers as implemented in the Python scikit-learn toolbox 0.14 [13]: naive Bayes and decision tree classifiers. We also tried k-nearest neighbor classifiers and stochastic gradient methods [14] but omit those results in this paper. We first describe the base classifiers and then discuss implementation of the cascaded classifier.

Naive Bayes Classifier. The first classifier we consider is naive Bayes, which applies Bayes' theorem with strong independence assumptions [14]. It has a conditional probability model $p(Y \mid X_1, \ldots, X_n)$, where Y is the dependent class that is conditional on the feature variables $X_1, \ldots, X_n$. Applying Bayes' Theorem and independence of the features, the conditional probability can be written as

$$p(Y \mid X_1, \ldots X_n) = \frac{1}{Z} p(Y) \prod_{i=1}^{n} p(X_i \mid Y), \quad (8)$$

where Z is a normalization constant. The estimated class $\widehat{y}$ given test data $x_1, \ldots, x_n$ is

$$\widehat{y} = \arg\max_y p(y) \prod_{i=1}^{n} p(x_i \mid y). \quad (9)$$

The class priors $p(y)$ are computed empirically from the training data; the independent probability distributions $p(x_i \mid y)$

are Gaussian with mean and variance estimated empirically from the training data. Although our features are not independent and the Gaussian model is not accurate for our model, naive Bayes is efficient in terms of CPU and memory and is a common testbench classifier in the literature [15].

Decision Tree Classifier. The Decision Tree classifier infers simple decision rules from features to build a tree and predict variable classes. Each node in the tree represents a type of feature, and each directed edge from that node refers to a particular instance of that feature. The classifier works by sorting each instance of the features down a tree from the root to a leaf node that gives the class of the instance [14]. We use the C4.5 decision tree algorithm [16].

Cascaded Classifier. After tuning, both base classifiers in our experiments yielded high detection rates but also high false alarm rates. For example, given activity vectors of 1000 churners and 1000 non-churners for each month of the 3.7 million caller network, training and testing a naive Bayes classifier with month 1 data yielded a 98% detection rate and a 44% false alarm rate for month 2. To reduce the false alarm rate, we build a cascaded classifier [5] for churner detection.

The cascaded classifier operates in stages. At each stage, we choose a feature category (see Section 3), a sample size M, and a base classifier that together minimize the false alarm rate while ensuring a detection rate above a given threshold. The vertices we classify as non-churners in the first stage are not considered in later stages and are discarded from the remaining test data. The second and subsequent stages are conceptually equivalent to stage 1. This process continues until the target false alarm rate is reached.

Suppose $p_{d,i}$ represents the detection rate of the ith stage, and $p_{f,i}$ denotes the false alarm rate of the ith stage, $i \in [1, L]$. Then the detection rate p_d of the cascaded classifier is the product of the L stage detection rates $p_{d,i}$. Likewise, the cascaded false alarm rate p_f is the product of the L stage false alarm rates $p_{f,i}$. Implementing these stages drastically reduces the false alarm rate. For example, a 4-stage cascade that has false alarm rate 0.4 at each stage will have an overall false alarm rate of 0.027. The detection rates also decrease at each stage, but hopefully by a lesser amount than the false alarm rate. We apply this classifer to obtain empirical results for the caller data in the next section.

	Naive Bayes		Decision Tree		Mixed Stages	
Stage	p_d	p_f	p_d	p_f	p_d	p_f
1	0.98	0.44	0.98	0.44	0.98	0.44
2	0.92	0.26	0.92	0.23	0.92	0.23
3	0.86	0.068	0.78	0.041	0.85	0.067
4	0.56	0.014	0.64	0.003	**0.71**	**0.008**

Table 1: Detection (p_d) and false alarm (p_f) rates for cascaded classifiers. Mixed stages yield the highest detection rate.

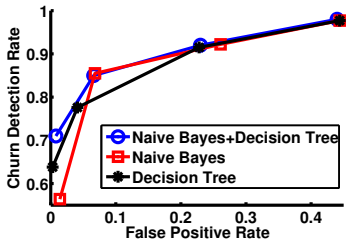

Fig. 2. Results for three cascaded classifiers. The circles (blue) show results for mixed naive Bayes and decision tree classifiers across stages. The squares (red) are for only naive Bayes and the asterisks (black) are for only decision tree classifiers across stages. We see that the subgraph activity difference features together with the seed vertex vectors and mixed stages allow a false alarm rate of 0.08% with 71% detection.

5. EMPIRICAL RESULTS

We consider two months of a 3.7 million caller dataset and construct networks as in Section 2. We uniformly sample seed vertices from each month so that 1000 churners and 1000 non-churners are collected. We use the method outlined in Sections 3 and 4 to compute features and train three cascaded classifiers using the month 1 data. Each classifier has an initial stage consisting of a naive Bayes classifier with the activity vectors of the seed vertices as features. The rest of the stages are either all naive Bayes classifiers, all decision trees, or a combination of both. After tuning, the decision trees have maximum depth 1 and use all features to find the best split.

Table 1 and Figure 2 illustrate our results. The mixed-stage classifier has the highest churn detection rate at 71%, while its false alarm rate is 0.8% – that is, we correctly identify 710 of 1000 month 2 churners while incorrectly classifying 8 of 1000 non-churners; to compare, the initial stage correctly identifies 980 of 1000 churners while incorrectly classifying 440 of 1000 non-churners.

6. CONCLUSION

We develop an efficient feature extraction and classification scheme for detecting churners in a 3.7 million caller network. We first construct features that quantify activity differences between connected subgraphs by computing integral affinity graphs. We then implement a cascaded classifier that yields 71% detection of the churners and a 0.8% false alarm rate. Our results illustrate that graph-dependent feature sets are a powerful tool that can be combined with single-vertex feature vectors to reduce the false alarm rate. In future work, we hope to improve the features and classifiers to achieve greater computational efficiency and further reduce the false alarm rate without compromising the detection rate.

7. ACKNOWLEDGEMENTS

We would like to thank i-Lab of Carnegie Mellon University for providing access to the mobile caller dataset.

8. REFERENCES

[1] J.L. de la Rosa, R. Mollet, M. Montaner, D. Ruiz, and V. Muñoz, "Kalman filters to generate customer behavior alarms," *Frontiers in Artificial Intelligence and Applications*, vol. 163, pp. 416–425, 2007.

[2] C. Archaux, A. Martin, and A. Khenchaf, "An SVM-based churn detector in prepaid mobile telephony," in *Proceedings of 2004 IEEE International Conference on Information and Communication Technologies: From Theory to Applications*, Apr. 2004, pp. 459–460.

[3] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos, "Neighborhood formation and anomaly detection in bipartite graphs," in *Proceedings of the 5th IEEE International Conference on Data Mining*, 2005, pp. 418–425.

[4] L. Akoglu, M. McGlohon, and C. Faloutsos, "Oddball: Spotting anomalies in weighted graphs," in *Advances in Knowledge Discovery and Data Mining, Lecture Notes in Computer Science*, vol. 6119, pp. 410–421. Springer Berlin Heidelberg, 2010.

[5] P. Viola and M.J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[6] L.A. Goodman, "Snowball sampling," *Annals of Mathematical Statistics*, vol. 32, no. 1, pp. 148–170, 1961.

[7] M. Gjoka, M. Kurant, C.T. Butts, and A. Markopoulou, "Walking in Facebook: A case study of unbiased sampling of OSNs," in *Proceedings of the 29th IEEE Conference on Computer Communications (INFOCOM)*, March 2010, pp. 1–9.

[8] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)*, 2006, pp. 631–636.

[9] M. Kurant, A. Markopoulou, and P. Thiran, "Towards unbiased BFS sampling," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1799 –1809, Oct. 2011.

[10] M.J. Salganik and D.D. Heckathorn, "Sampling and estimation in hidden populations using respondent-driven sampling," *Sociological Methodology*, vol. 34, pp. 193–239, 2004.

[11] J.A. Deri and J.M.F. Moura, "Graph sampling: Estimation of degree distributions," in *Proceedings of the 38th IEEE International Conference on Acoustics, Speech, and Signal Processing*, May 2013.

[12] T.H. Cormen, C. Stein, R.L. Rivest, and C.E. Leiserson, *Introduction to Algorithms*, McGraw-Hill Higher Education, 2nd edition, 2001.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] T.M. Mitchell, *Machine Learning*, McGraw-Hill, Inc., New York, NY, USA, 1997.

[15] H. Zhang, "The optimality of naive Bayes," in *Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference*, May 2004, pp. 562–567.

[16] J.R. Quinlan, *C4.5: Programs for Machine Learning*, vol. 1, Morgan Kaufmann, 1993.