Fraud Detection: Methods of Analysis for Hypergraph Data

Anna Leontjeva, Konstantin Tretyakov, Jaak Vilo
Department of Computer Science
University of Tartu
Tartu, Estonia
Email: {anna.leontjeva, kt, jaak.vilo}@ut.ee

Taavi Tamkivi
Department of Mathematical Statistics
University of Tartu
Tartu, Estonia
Email: taavi.tamkivi@ut.ee

***Abstract*—Hypergraph is a data structure that captures many-to-many relations. It comes up in various contexts, one of those being the task of detecting fraudulent users of an on-line system given known associations between the users and types of activities they take part in. In this work we explore three approaches for applying general-purpose machine learning methods to such data. We evaluate the proposed approaches on a real-life dataset of customers and achieve promising results.**

I. Introduction

An intense growth in the volume of available scientific and commercial data is one of the hallmarks of the modern age. Despite the similarly fast progress in the data analysis methods, their use with discrete data, such as text and relations, is often not straightforward and depends highly on application.

This work focuses on the classification of data structured as a *hypergraph*. A hypergraph is data structure, encompassing a set of items (*vertices*) and a set of relations (*hyperedges*) among those items, such that each relation may encompass any number of objects. This is in contrast to a usual graph, where relations are always binary. A hypergraph can be represented as a bipartite graph.

In general, the structure of hypergraph is used in the context of a multi-label problem, where each instance can belong to a set of labels. The most common example of a hypergraph is the data about articles and their authors [1]. Several people can be involved in writing the same article, as well as several articles can have the same author. We can thus refer to authors as *vertices* and co-authored publications as *hyperedges* of a hypergraph.

Other examples of hypergraph can be seen in such applications as text categorization [2], protein function prediction [3] and image classification [4]. In addition, hypergraphs have been applied in the context of spectral graph-theoretical methods [5], [6].

The hypergraph considered in this work is based on the data of a certain internet voice call company, who is interested in detecting fraudulent customers. The company has information about the activities of its users, such as the IP-addresses from which they connect, phone numbers they call, e-mail they use, etc. The relationship between users and particular IP-addresses, phone numbers and e-mails is many-to-many. Hence, the data is a hypergraph with users corresponding to vertices and activities to hyperedges. To be more precise, each type of activity forms a separate hypergraph, or, as we denote it here, a separate *color*. Thus, we are dealing with a *colored hypergraph*. The similar problem has been studied in [7], where the anomalies of the data were detected using unsupervised techniques.

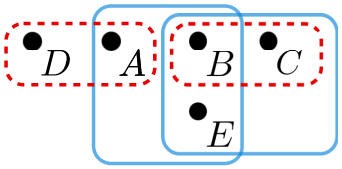

Fig. 1. Example of a colored hypergraph with 5 vertices, four hyperedges and two colors.

Our objective is to learn a classifier for vertices in a colored hypergraph which would primarily take into account the hypergraph structure of a vertex's neighborhood. The premise is that such an approach could beneficially complement existing classical approaches, such as simply blacklisting "bad" IP-addresses. We approach this task by applying generic *distance-based*, *kernel-based* and *feature-vector-based* techniques to learn a classifier for vertex neighborhoods (which, themselves, are small hypergraphs).

The obtained results are quite satisfactory given the particular task and data available.

A. Basic definitions

The main focus of this paper is the notion of a *hypergraph*.

Definition 1: A *hypergraph* $\mathbb{H}$ is a pair $\mathbb{H} = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ is a set of *vertices*, and $\mathbb{E}$ is a (multi-)set of non-empty subsets of $\mathbb{V}$, called *hyperedges*.

If we assign discrete labels to hyperedges, we obtain a *colored hypergraph* (see Figure 1).

Definition 2: A *colored hypergraph* is a tuple $\mathbb{H} = (\mathbb{V}, \mathbb{E}, \mathbb{C}, c)$, where $(\mathbb{V}, \mathbb{E})$ is a hypergraph, $\mathbb{C}$ is a set of colors, and the function $c : \mathbb{E} \to \mathbb{C}$ specifies a color for each hyperedge.

We call two vertices v_i and v_j *neighbors*, if there is an edge connecting them.

Definition 3: For a hypergraph $\mathbb{H} = (\mathbb{V}, \mathbb{E})$ the *neighborhood* of a vertex v in a hypergraph $\mathbb{H}$, $N(v_i)$, is the set

consisting of all vertices adjacent to v, i.e.

$$N(v_i) = \{v_j \in \mathbb{V}, \exists e \in \mathbb{E} : \{v_i, v_j\} \subset e\}.$$

In the following we shall also use the term *neighborhood* to refer to the sub-hypergraph, induced by $N(v_i)$.

II. Methods and materials

A. Hypergraph classification problem

Consider a colored hypergraph $\mathbb{H} = (\mathbb{V}, \mathbb{E}, \mathbb{C}, c)$ as defined in Definition 2. Let there be a (possibly stochastic) function

$$f : \mathbb{V} \to \{\text{Positive}, \text{Negative}\},$$

defining a class for each vertex. Suppose that we observe the values of $f(v)$ for a subset of vertices $\mathbb{V}_s$. The task of *hypergraph vertex classifier learning* is to estimate a predictive model $\hat{f}$, which could predict the classes (or their probabilities) of any vertex. In this work, we are only interested in models $\hat{f}$, which classify a given vertex v by considering only its *neighborhood* – a subgraph formed on a set of nearby vertices. This is a common practical limitation, related to the way user activity data can be accessed. However, the neighborhood of a hypergraph vertex is itself a hypergraph. Thus, a model, which classifies *vertices* on the basis of their neighborhoods is essentially a *hypergraph classifier*.

Definition 4: Suppose we are given a training sample $D = \{(h_i, y_i)\}$ of colored hypergraphs ($h_i \in \mathcal{H}$) with the corresponding class labels ($y_i \sim f(h_i)$). The task of *hypergraph classifier learning* is to estimate a predictive model $\hat{f}$, generalizing the relationship present in the sample and thus approximating the true f.

The complication of the problem lies in the fact that the predictors are hypergraphs, rather than simple numeric vectors. There are no standard recipes for handling hypergraph inputs. Indeed, in the most general case, any two hypergraphs may be considered as two distinct discrete inputs. Naturally, this view is rather useless in practice. In order to be able to generalize from the available information to future data we need to exploit the structure of the hypergraph and explicitly represent it in the model. Here we attempt to apply instances of *distance-based*, *kernel-based* [8], and a *feature vector representation* to solve the problem.

B. Distance-based approach

A *distance* is a function that measures the dissimilarity of two objects.

Definition 5: A *distance function* on the set X is a function $d : X \times X \to \mathbb{R}$, which satisfies:

$$\begin{aligned} &d(x, y) = 0 \Leftrightarrow x = y \\ &d(x, y) = d(y, x), \text{ for all } x, y \in X, \\ &d(x, z) \leq d(x, y) + d(y, z) \text{ for all } x, y, z \in X. \end{aligned}$$

Once a distance function is defined for a given set X, it becomes possible to solve learning tasks involving elements of X. The *K-nearest-neighbors* is the most straightforward classifier learning algorithm in this area.

Definition 6: Let X be a set of objects and let d be a distance function defined on X. Let $D = \{(x_i, y_i)\}$ be a dataset of observations ($x_i \in X$), labeled by their class ($y_i \in \{-1, 1\}$). The K-nearest neighbors classifier classifies new observations according to the majority class among the K nearest neighbors of a given instance:

$$\text{KNN}(x) = \text{sign}\left(\sum_{i \in \text{Nearest}_k(x, D)} y_i\right),$$

where $\text{Nearest}_k(x, D)$ denotes the set of k nearest neighbors of x in the dataset D according to distance d.

In order to apply a distance-based method to our problem we need to define a distance function on hypergraphs. The most natural candidates are based on the idea of counting edit operations that are necessary to turn one graph into another. Unfortunately, such approaches are computationally too demanding to be useful in practice. In order to be able to apply the distance-based approach for the practical part of this work we propose a simplified approximation.

Definition 7: Let $\mathbb{H}_1$ and $\mathbb{H}_2$ be two colored hypergraphs. For each color c and each hypergraph $\mathbb{H}_i$ ($i \in \{1, 2\}$), order the c-colored hyperedges of $\mathbb{H}_i$ by their size. Let s^c_{i1} be the size of the largest hyperedge of color c, s^ℓ_{i2} the size of the second largest hyperedge, etc. Suppose there are N^c_i hyperedges of color c in hypergraph i, and let $s^c_{ij} = 0$ for $j > N^c_i$. Define the *simplified hypergraph distance* between $\mathbb{H}_1$ and $\mathbb{H}_2$ as:

$$d(\mathbb{H}_1, \mathbb{H}_2) = \sum_c \sum_{j=1}^{\infty} |s^c_{1j} - s^c_{2j}|.$$

C. Kernel-based approach

A similar approach to structured data classification learning is based on kernel methods [9], [10]. Analogously to a distance function, a kernel is a measure of similarity.

Definition 8: A function $K : X \times X \to \mathbb{R}$, is a *kernel* if there exists a Hilbert space H and a mapping $\phi : X \to H$, such that $K(x, y) = \langle \phi(x), \phi(y) \rangle_H$.

When a kernel function is available, it becomes possible to learn *linear classifiers in dual form*, such as Support Vector Machines (SVMs).

Definition 9: A kernel-based linear classifier is a function

$$\text{KC}(x) = \text{sign}\left(\alpha_0 + \sum_i \alpha_i y_i K(x, x_i)\right)$$

where K is a kernel function, $D = \{(x_i, y_i)\}$ is a training set as in Definition 6, and α_i is a set of model parameters to be estimated from the data D.

In particular, in our experiments we use the SVM parameter learning algorithm.

Various kernels have been proposed for structured data. In particular, a considerable amount of research has been done on graph kernels [11], [12], [13]. Wachman [14] introduced a general kernel for colored hypergraphs based on random walks. Unfortunately it turned out to be computationally too demanding for our purposes and we have devised a *simplified*

hypergraph kernel, which is both fairly easy to compute efficiently, and relevant for representing neighborhood hypergraphs for the purpose of fraud detection.

Definition 10: Define the *type* $\tau(v)$ of a vertex v in a colored hypergraph $\mathbb{H}$ to be the multiset of colors of edges, which contain this vertex:

$$\tau(v) = \{c(e),\, e \in \mathbb{E}, v \subset e\}.$$

Definition 11: For a colored hypergraph $\mathbb{H}$ and a vertex type τ define by $\#(\mathbb{H}, \tau)$ the number of vertices of type τ present in the hypergraph $\mathbb{H}$. The *simplified hypergraph kernel* is defined as:

$$K_h(\mathbb{H}_1, \mathbb{H}_2) = \sum_{\tau \text{ is a vertex type}} \#(\mathbb{H}_1, \tau)\#(\mathbb{H}_2, \tau)\,.$$

The simple hypergraph kernel can be extended using standard constructions [10] to obtain *polynomial* and *RBF* versions.

Definition 12: Define a *degree d polynomial hypergraph kernel as*

$$K_{hpoly}(\mathbb{H}_1, \mathbb{H}_2) = (K_h(\mathbb{H}_1, \mathbb{H}_2) + 1)^d.$$

Definition 13: Define a *RBF hypergraph kernel* as

$$K_{hrbf}(\mathbb{H}_1, \mathbb{H}_2) = \exp\left(-\gamma \sum_{\tau} (\#(\mathbb{H}_1, \tau) - \#(\mathbb{H}_2, \tau))^2\right),$$

where the γ is a tunable parameter.

D. Feature vector approach

The last and the most straightforward approach to learning on structured data is simply to convert it to a fixed-length vector of features. This allows to apply nearly any state of the art learning algorithm. In order to examine a simpler alternative to our distance- and kernel-based representations, in our experiment we consider a very basic representation of a colored hypergraph in terms of m features, where m is the number of colors. Each feature $\phi_k(\mathbb{H})$ simply counts the number of vertices belonging to at least one edge of color k:

$$\phi_k(\mathbb{H}) = |\{v \in \mathbb{V} \,|\, \exists e \in \mathbb{E} : v \in e, c(e) = k\}|.$$

We use *linear classifier learning* techniques (*logistic regression* and SVM)with such feature representation in our experiment.

Definition 14: Given a feature representation

$$x \to (\phi_1(x), \phi_2(x), \ldots, \phi_m(x)),$$

a *linear classifier* is a function of the form

$$LC(x) = w_0 + \sum_{j=1}^{m} w_j \phi_j(x)\,,$$

where w_i are model parameters to be estimated from the data.

E. Example

For clarity, let us illustrate the presented approaches using the graph on Figure 1. Assume the hypergraph depicts a set of five customers $\{A,B,C,D,E\}$ and two kinds of activities (e.g. phone calls, corresponding to dashed hyperedges and emails, corresponding to solid ones). Thus, customers A and D are known to have called the same phone and customers A,B and E used the same e-mail in their registration data.

Consider customer A. In our application we shall only observe its immediate neighborhood, which consists of users $\{A,B,D,E\}$ joined to A via two hyperedges. For customer B the immediate neighborhood encompasses users $\{A,B,C,E\}$ joined by three hyperedges.

a) Approximate hypergraph distance: For A's neighborhood, the sequence of hyperedge sizes is $(3, 0, \ldots)$ for the "solid" type and $(2, 0, \ldots)$ for the "dashed" type. Edge size sequences for B's neighborhood hypergraph is $(3, 3, 0, \ldots)$ and $(2, 0, \ldots)$ correspondingly. Consequently, the simplified hypergraph distance between A and B neighborhoods is $d(A, B) = |3 - 3| + |0 - 3| + |2 - 2| = 3$.

b) Simple hypergraph kernel: The neighborhood of A has 1 vertex of type {dashed}, 1 vertex of type {dashed,solid} and 2 vertices of type {solid}. The neighborhood of B has 1 vertex of type {solid}, 1 vertex of type {solid, solid}, 1 vertex of type {dashed, solid} and 1 vertex of type {dashed,solid, solid}. There is thus just one common vertex type {dashed, solid} and the value of the simplified kernel is $K_h(A, B) = 1 \cdot 1 = 1$.

c) Feature representation: As there are two colors, representation will be two-dimensional. The neighborhood of A has $\phi(A) = (3, 2)$ and the neighborhood of B has $\phi(B) = (4, 2)$.

III. Experimental Evaluation

For evaluation we applied our methods on a sample of users of an internet voice company, testing the performance of the suggested techniques for fraud detection.

A. Dataset

The data comprises a sample of users of the communication network together with a list of different activities of 11 kinds. Each activity includes a list of users, who were recently involved in it. For example:

```
IP=80.1.2.3      { user1, user2, user3 }
IP=80.1.2.5      {       user2, user3 }
Phone=555-10-20  { user1,       user3 }
```

Here users `user1`, `user2`, `user3` have recently (e.g. within the last week) been known to connect from IP address `80.1.2.3`. Analogously, users `user1` and `user3` both dialed phone number 555-10-20 during the last week.

We precompute one-step hypergraph neighborhoods for each user and store them in the way convenient for further computations of the methods described.

Each user has an associated class, indicating whether he is known to be fraudulent or not. The class distribution is highly skewed, with the proportion of fraudulent users being less than

0.5%. To enable learning, we create a *balanced* training set by selecting an equal number of both fraudulent and honest users. As a result we obtain a training set of size up to 6000.

B. Results

We compare the following algorithms:

- *Logistic regression* classifier learning algorithm together with the feature vector representation.
- *SVM* algorithm with three different kernels (linear, polynomial of degree $d = 2$, RBF with γ tuned using cross-validation) together with both the feature vector representation and the proposed hypergraph kernel.
- *K-nearest neighbors* algorithm with $k = 1$ (as it provided the best results) together with the proposed distance function.

As already mentioned, we use a *balanced* training set to train the algorithms. We use a similarly balanced test set of the same size for estimating *accuracy* (proportion of correctly classified users) and *ROC AUC score* (the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one). These measurements are presented in Table I below. Note that we cannot compute *ROC AUC score* for the *K-nearest neighbors* method, as it does not have a parameter for trading precision for recall.

Apart from the bad performance of the KNN algorithm with our distance function, the differences in accuracy between the other algorithms are rather small.

	Simple		Kernel	
Model	**Accuracy**	**AUC**	**Accuracy**	**AUC**
Logit	0.71	0.84	-	-
SVM_linear	0.70	0.82	0.76	0.83
SVM_poly	0.65	0.75	0.66	0.80
SVM_rbf	0.78	0.84	0.77	0.83
K-NN	0.55	-	-	-

TABLE I
ACCURACY AND ROC AUC SCORE

A practically more relevant metric is *precision at 99%*, computed on an unbalanced (skewed) data (Table II). It directly measures the proportion of fraudulent users which could be discovered by the algorithm while having a false alarm rate less than 1%. We observe that the results are much more modest and dispersed, with the simpler approaches showing best performance.

Examining the actual ROC curves (Figure 2) we can confirm that despite having nearly equal AUC, some of the curves have a steeper ascent rate at the leftmost (low false-positive) region.

Model	**Simple Recall at 99**	**Kernel Recall at 99**
Logit	0.16	-
SVM_linear	0.15	0.09
SVM_poly	0.09	0.01
SVM_rbf	0.04	0.05

TABLE II
RECALL AT A FIXED PRECISION OF 99%

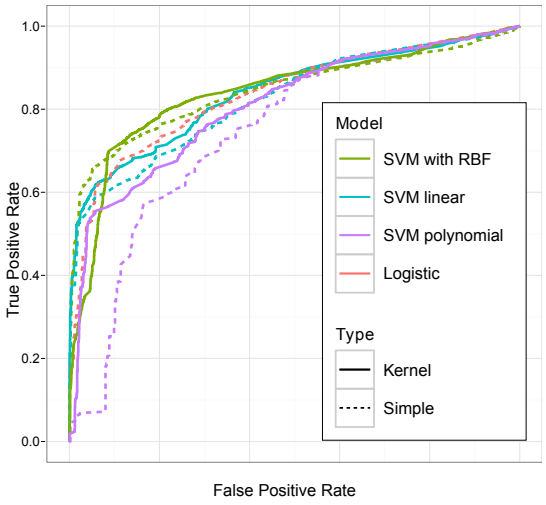

Fig. 2. ROC curve comparison of different algorithms

IV. CONCLUSION

We presented an instance of a fraud detection problem, framed as a hypergraph classifier learning task. We described three generic approaches for tackling the problem and applied them on the data of an internet voice call company. Our findings show that a system with a recall of at least 16% at a desirably high precision value can be created. Such a system, although not perfect on its own, may be used to enhance the recall of existing fraud detection algorithms, which do not take the hypergraph structure into account.

REFERENCES

[1] M. Ley and P. Reuther, "Maintaining an online bibliographical database: the problem of data quality. in egc, ser. revue des nouvelles technologies de l' information, vol. rnti-e-6," *Cépadués Éditions*, vol. 2006, pp. 5–10, 2006.

[2] S. Yu, K. Yu, V. Tresp, and H.-P. Kriegel, "Multi-output regularized feature projection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, pp. 1600–1613, 2006.

[3] Z. Tian, T. Hwang, and R. Kuang, "A hypergraph-based learning algorithm for classifying gene expression and arraycgh data with prior knowledge," 2009. [Online]. Available: http://bioinformatics.oxfordjournals.org/cgi/content/short/25/21/2831

[4] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification," 2004.

[5] L. Sun, S. Ji, and J. Ye, "Hypergraph spectral learning for multi-label classification," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '08. New York, NY, USA: ACM, 2008, pp. 668–676. [Online]. Available: http://doi.acm.org/10.1145/1401890.1401971

[6] G. Chen, J. Zhang, F. Wang, C. Zhang, and Y. Gao, "Efficient multi-label classification with hypergraph regularization," *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, vol. 0, pp. 1658–1665, 2009.

[7] J. Silva and R. Willett, "Hypergraph-based anomaly detection of high-dimensional co-occurrences," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, pp. 563–569, 2009.

[8] T. Gärtner, J. W. Lloyd, and P. A. Flach, "Kernels and distances for structured data," *Machine Learning*, vol. Volume 57, pp. 205–232, 2004. [Online]. Available: http://www.springerlink.com/content/P101U19064697034

[9] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.

[10] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. New York, NY, USA: Cambridge University Press, 2004.

[11] T. Gärtner, "Predictive graph mining with kernel methods," in *Advanced Methods for Knowledge Discovery from Complex Data*, 2005.

[12] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized kernels between labeled graphs," in *Proceedings of the Twentieth International Conference on Machine Learning*. AAAI Press, 2003, pp. 321–328.

[13] T. Horváth, T. Gärtner, and S. Wrobel, "Cyclic pattern kernels for predictive graph mining," in *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2004, pp. 158–167.

[14] G. M. Wachman, "Kernel methods and their application to structured data," Tufts University, Tech. Rep., 2009.