General Expert System Anomaly Detection for Streaming Data

Andrew Butcher, Oussama Elrawas, Ekrem Kocagüneli
Lane Department of Computer Science and Electrical Engineering
West Virginia University, Morgantown, WV 26506
abutcher@afrolegs.com, orawas@gmail.com, ekocagun@mix.wvu.edu

***Abstract*—The effort to model and understand knowledge and data has given rise to a large variety of implementations of knowledge level modeling. Aimed at achieving various operations such as anomaly detection, classification and planning, among others, knowledge level modeling allows us to derive generalizations concerning data regardless of the nature of the data. In this paper, we present a preliminary toolkit aimed conducting KL modeling on a continuous stream of data. This toolkit will be presented at this stage of its implementation to allow for anomaly detection in the data stream.**

I. Introduction

Expert systems have been proposed to solve various problems in many different contexts. Although these systems were developed separately from each other, all of them share common properties. The realization of the properties have led to an exciting idea of *knowledge level modeling* (KL). Knowledge level modeling aims to find abstract patterns of inference that appear in various expert systems [3].

In fact the idea of KL is not a very new concept. In 70s and 80s, some applications of high level expert systems have been reported. Although some research was conducted to make this initial work more common and widely applicable, the current trend of expert system design usually consists of a somewhat trial and error approach [2].

Although trial and error method when combined with expert domain knowledge may yield very successful results, most of these models require quite a lot of planning, design and research prior to implementation. Therefore, the failure to exploit re-usable abstract domain independent problem solving strategies result in waste of resources. Previous research has also adressed this issue and reported three benefits of using KL modeling [3]:

- Reuse Benefit:New designs can be bootstratpped from previous designs. The new design does not need to be a copy of the previous one and may introduce various new configurations. However, essential pattern will be reused and be the start point of a new design.
- Communication Benefit: KL models could be very useful to explain various different expert systems and a novice designer with the knowledge of KL models could easily adapt to a new expert system.
- Guidance Benefit: By analyzing previous models, designers could get an insight regarding where to direct their focus in the design of a new expert system.

The fundamental idea behind KL is that a knowledge base is divided into two parts: 1) Domain specific facts and 2) domain independent problem solving strategies [2]. However, a wide range of current expert system designs are not based on this fundamental idea and are missing the afore mentioned benefits provided by KL modeling.

In this research, we are making an analysis of previous KL models and going further we are proposing a common toolbox that includes common methods or tools to these models. After our analysis, we will build an expert system that will make use of previous KL models and the necessary methods from the common toolbox of KL models. The model will be applied to a world that is perceived by streaming data, which corresponds to a generalized version of part one for a knowledge base, i.e. a general representation for domain specific facts. This will enable model to be applicable to any specific domain that can be mapped to the described representative model.

By building this model on the principles of KL modeling, we do not only exploit the benefits of KL modeling but we also propose a widely applicable model for a large set of domains.

II. background

In this section WE will describe and briefly discuss previous work related to the area of knowledge level modelling. As discussed in the introduction, KL modelling is well researched concept and there have been several variations on the theme. KL modelling can be presented in one of two different types of studies and implementations:

- General all encumpassing work on knowledge engineering that attempt to cover and implement a wide range of inference functions. These implementation generally do not target a specific type of data.
- Specific work on KL modeling that seek to implement a small subset of functionality. Usually such implementations aim at targeting specific data sets.

The more encompassing studies seek to implement extensive KL functionality that is able derive knowledge from most data that is available to their system. such work include the modeling of cognitive processes that is presented in [1] by Clancey et. al. In this publication, the authors present flowchart style description to several processes that include:

-

- Diagnosis

- Verification
- correlation
- suitability
- classification
- Prediction
- Repair
- Design
- Configuration
- Planning
- Scheduling

While implementations to these descriptions are not implemented, it is important to take note of them. Such extensive descriptions represent the tradition of knowledge modeling by way of extensive specification of any knowledge/theory producing process. These processes are briefly explained in following sections. As we will see, our own tool is a is based on his tradition of specification, or over-specification according to who you ask [5].

Detailing current work regarding this approach to knowledge modeling is currently out of the scope of this paper. However, we will briefly present two papers that lie on either side of the fence of this knowledge modeling equation.

The first paper [2] by Menzies provides a description and a system (HT4) that follows the above tradition and which is done through abduction. Abduction here meaning that rules (or hypotheses) are produced such that, given the final effect/state, we are able to most closely determine our initial effect/state. in other words, we are producing rules that will allow us to produce/ infer old data given our current data. This method is obviously dependent on observing enough data to produce out rules. These rules will ultimately define our knowledge. In a similar manner to the cognitive processes document, this work attempts to apply knowledge modeling to achieve several of the functions mentioned above. Such functions include prediction, classification, explanation, tutoring, planning, monitoring, validation, verification and diagnosis. While not identical to the list of functions mentioned above, it overlaps with regard to most of the functionality. As such, the author presents a general tool for use in KL. Our proposed future implementation will be similar to this, with the main difference being that our method is based on induction.

While this is one side of knowledge modeling, the other side of knowledge modeling forgoes specifying all the above functionality, and instead specifies one rule: remember the past to determine the present [5]. This is called Case Based Reasoning (CBR). Argued for by Riesbeck, this method represents knowledge and experience with dealing with that knowledge in the form of case bases that include historical data and actions performed on the data in the past. Current actions are determined by the similarity of our current case/data to previous data. This method is based on the principle that people don't create new decisions based on cognitive analysis, but rather that current actions are based on previous actions conducted in similar situations.

In the next section we will present a description of the model that we will be using.

A. Library and Toolbox For Design Patterns

KL patterns appear in many different expert systems. In that respect the goal of KL modeling can be defined as tpeo identify the abstract reusable inference skeletons that appear multiple times in such systems. A very good example to this concept is the example of Clancey [1], where he reverse engineered 10 expert system tools with various different characteristics and found out that all of them were using the same abstract inference skeleton: Heuristic classification. Of course Clancey's example is not the only one. Another KL methodology KADS [6] was reported to have been used in more than 40 knowledge-based systems [3].

Bearing in mind that KL inferece models are used repeatedly in many different expert systems, it is a good idea to collect and store them. Clancey proposes a library of pre-defined problem solving strategies and a a separate knowledge-base that contains domain specific heuristics [6]. In this library we can define common problem solving models that underlie all the expert systems. When the common problem solving strategy is combined with separate domain specific heuristics, then an expert system can easily be built with less effort. As examples of models in such a library we can name verification, incremental configuration, systematic diagnosis and heuristic classification.

Going further we would also like to mention a toolbox for such a library that is proposed by Menzies [4]. When we take a closer look to the library proposed by Clancey [6] we see that various problem solving models make use of common methods or algorithms. For instance verification, classification, sampling methods, diagnosis, discretization, median and greedy agglomerative clustering (GAC) are examples to common methods in the toolbox that can be used by various inference skeletons to form multiple expert systems in different settings. We can in fact group these algorithms in a toolbox that is available to the use of models in the library of Clancey.

So we can think of building an expert system as a three step process:

1) Choose a model/template best for your case from the library
2) Pick up required methods from toolbox
3) Insert domain specific heuristics

In our study we will follow the above described three steps to build our model that is capable of processing continuous data and detecting anomalies. Since we are aiming to provide a generic model, we will not have any domain specific heuristics in our work. Therefore, we will cover only the first two steps while building our model.

B. Greedy Agglomerative Clustering

Since the fundamental method in our model will be the greedy agglomerative clustering (GAC), we will first provide some information regarding GAC. Agglomerative, or bottom up, clustering starts with every individual data point and greedily combines similar points using a distance metric [**?**]. Our model uses euclidean distance to pair similar data points.

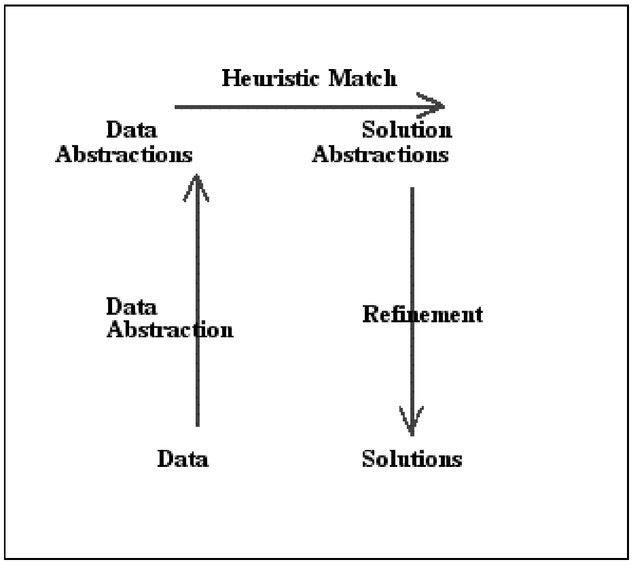

Fig. 1: Heuristic classification takes raw data and applies and abstraction. Then on the data abstraction heuristic match methods are run, which in our case is a distance based GAC. Then hypothesis coming from the heuristic match is evaluated and the solution is reached.

The clusters formed can then be mapped into a tree structure. A GAC tree partitions the data such that the leaves in the tree are the original data points and each interior node is a cluster containing every point in its subtree. The root thus contains every data point.

When querying the tree we can easily throw away whole subtrees by comparing our test with the cluster at hand. This speeds up traversal greatly, allowing for sub-linear query costs.

III. Our Model

Our model processes streaming data in chunks of certain number of instances and tries to find anomalies within the processed data. For learning what is abnormal and what is normal, the algorithm relies on past data and tries to For our case, we use 100 instances in each chunk, since GAC may have performance drawbacks with datasets that have more than 100 instances. While building our model, we will use the three step procedure that was described in Section II-A. In the following subsections we will describe these steps.

A. Select a template from the library

Since our aim is to find anomalies with this model, we are in fact dealing with a classification problem where we have two classes: Normal and abnormal. The most suitable template for the type of problem we are tryinig to tackle with is ”heuristic classification”.

```
divide dataset into chunks of 100 and process each chunk at a time
for each chunk run 7 times
randomize training dataset
instancei = pick up an instance from in chunki
clusters = build a GAC tree from chunki minux instancei instances
centroidj = mean(clusterj) for all clusters in GAC tree
calculate the distance of of instancei to all the centroids
find closest centroid and see its level in the tree
if level of closest centroid ¡ floor( # of all levels in the tree / 2)
then give normal class label to instancei
otherwise give abnormal class label to instancei
```

Fig. 2: Pseudocode for training of generic GAC anomaly detector

B. Pick up methods from toolbox

Now we have a basic frame to build our model, but we need to select the appropriate tools to be able to implement this model. To abstract our data we expect the data to be provided to the model in the form of a two dimensional array in which the rows will correspond to instances and the columns will correspond to features of the instances. Then the data will be normalized to the interval of 0 to 1 and this will form our abstracted database. Therefore, we will use a ***“normalize”*** method from the toolbox first.

Once the data is normalized, the heuristic match mechanism will start to work. Since we are building a GAC tree on the data, we will need a ***“GAC”*** method and we will need to calculate the distances between clusters. Thus, we will pick up a ***“distance”*** function from the toolbox as well. We calculate the distance of a single instance to the centroid of clusters in GAC, and centroids are represented by the mean of all the features of all the instances in that cluster. Therefore, we will also select ***“mean”*** method of the toolbox.

We now have the template and the required tools to build our expert system. The third step would be to insert domain specific heuristics into the model. However, the model does not target any domain for the time being, therefore we will not include this step.

The model whose specifications have been verbally given above is a generic GAC anomaly detector. For more detail, we provide the pseudocode in Figure 2:

The logic here is that, we want instancei to be closer to majority of the instances in a tree and not close to leaves, which represent single/specific cases. If instancei is closer to specific instances than to a majority, then it is more likely to an anomaly. We do the above loop until we process all our dataset. At the end we have instances at our hand, which make up a normal world according to our definition.

After the training, we will have 7 labels for each instance in the chunk of 100 instances. Depending on the majority of the labels, decide whether an instance is normal or abnormal. Then throw away all the abnormals. Pick up another chunk and apply the same algorithm We do the above loop until we process all our dataset. At the end we have instances at our hand, which make up a normal world according to our definition.

For testing, use a similar strategy. But this time use the

```
randomize our normal world and then make chunks of 100
build a GAC on chunki
calculate distance of every test instance to centroids and label them in the same manner to training
after building a GAC for every chunk and labeling test instances for each chunk, we will have training-size/100 labels for each test instance
depending on the percentage of labels, we have 3 options for each test instance:
decide its normal (if 75% of the labels indicate its normal)
decide its abnormal (if 75% of labels indicate its abnormal)
decide that you cannot make a sound decision (if none of the labels exceed 75%)
```

Fig. 3: Pseudocode for testing of generic GAC anomaly detector

normal instances that were found before for training. The pseudocode for testing is given in Figure 3.

IV. Conclusion

The conclusion goes here.

V. Acknowledgment

We would like to thank...

References

[1] W. J. Clancey. Heuristic classification. *Artif. Intell.*, 27(3):289–350, 1985.

[2] T. Menzies. Applications of abduction: knowledge-level modelling. *Int. J. Hum.-Comput. Stud.*, 45(3):305–335, 1996.

[3] T. Menzies. Object-oriented patterns: Lessons from expert systems, 1997.

[4] T. Menzies. Theory of everything: Toe, 2009.

[5] C. Riesbeck. What Next? The Future of Case-based reasoning in Post-Modem AI. . *In Leake, ed., Case- Based Reasoning: Experiences, Lessons, and Future Directions.*, 1996.

[6] B. J. Wielinga, A. T. Schreiber, and J. A. Breuker. Kads: a modelling approach to knowledge engineering. *Knowl. Acquis.*, 4(1):5–53, 1992.