General Expert System for Anomaly Detection

Andrew Butcher, Oussama Elrawas, Ekrem Kocaguneli
Lane Department of Computer Science and Electrical Engineering
West Virginia University
Morgantown, WV 26505, USA
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. In this paper, we present a toolkit aimed conducting KL modeling. This toolkit will be presented at this stage of its implementation to allow for anomaly detection in classification data. By using this toolkit, we implemented a two-step, likelihood-based anomaly detector and tested it on simulated classification datasets for various different scenarios. Our model has achieved to perfectly identify the normal and abnormal test instances from the simulated datasets.

1. INTRODUCTION

Expert systems have been proposed to solve various problems in many different contexts [8, 11]. 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 [5, 7, 9]. 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 classification dataset and its simulated subsets, which correspond to a generalized version of part one for a knowledge base, i.e. a general representation for domain specific facts.

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.

2. 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 implementations 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 [6].

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.

When one side of knowledge modeling is that, the other side of knowledge modeling forgoes specifying all the above functionality, and instead specifies one rule: remember the past to determine the present [6]. 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.

2.1 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 to 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 [10] 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 [10]. 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 [10] 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.

3. METHODOLOGY

3.1 Datasets

In this research we are using 10 different classification datasets with different properties. The details regarding these datasets are as follows:

kr-vs-kp: This is a completely discrete dataset that describes a Chess end game. In this case it is black King and Pawn, the latter located on A7, versus white King and Rook. Each instance describes the positions of the chess peices on the board, where the class attribute indicates whether white can win or not. This dataset is also a fairly large dataset that is able to test the performance of our data generation routines. There are 37 attributes total and 3196 instances in this dataset.

mushroom: This is a completely discrete dataset that offers a large amoutn of instances of mushroom descriptions regarding the different parts of the mushroom. The classification of each instance indicates whether the mushroom is edible or poisonous. There are 23 attributes total and 8124 instances in this dataset, 2480 of which have missing attributes.

ionosphere: This is a medium size dataset that shows radar data from a system in Goose Bay, Labrador studying the ionosphere. All the attributes in this dataset are continuous, except for the class attribute. There are two classes of radar signals indicated: "good", those that are reflected and able to show ionoshpere structure, and "bad", those that go straight through. There are 35 attributes total and 351 instances in this dataset.

german-credit: This dataset shows anonymized credit data from germany. This includes data on the credit itself (eg. installment rate), and data on the person holding the credit (eg. status and sex). Each credit situation is classified as either "good" or "bad". This dataset has a mix of attribute types, with 7 continuous and 14 discrete attributes, including the class attribute. It contains 1000 instances.

credit-rating: This dataset has similar characteristics to the german credit dataset, consisting of a mix of continuous and discrete attributes. The instances describe credit card applications, where all the data has been anonymized. There are two classes: "+" and "-". The dataset contains 6 continuous and 10 discrete attributes, including the class, with 690 instances total, 37 of which have missing values.

cpu: This is a dataset consisting of mainly continuous (8) attributes describing characteristics of cpus of different vendors. The vendor name is the main discrete attribute in this dataset, and it will be used to represent the class. This makes this dataset a multiclass dataset, with 30 unique classes. In total, there are 209 instances in this dataset.

contactlens: This is one of the smaller datasets we are using, and contains 3 classes describing different scenarios for prescribing different contact lenses. The three classes are "hard", "soft", and "none". The attributes included in this dataset are all discrete, with 5 attributes total, including the class attribute. There are 24 instances total.

cloud: With 194 instances, this dataset includes 2 discrete and 5 continuous attributes. This dataset shows data related to a cloud seeding experiment in Tasmania. The season attribute is used as the class attribute in our experiment.

cleveland-14-heart-disease: This dataset is a 14 attribute version of the Cleveland heart disease dataset. The class attribute that will be used is the "cp" attribute, which has four unique values. Of the 14 attributes, 6 are continuous, with the rest being discrete. In total, this dataset has 303 instances.

breast-cancer: The breast cancer dataset shows instances of recurrence and no recurrence of cancer in breast cancer patients. The class attribute used is this recurrence attribute, which, along with all the attributes here, is discrete. There are 10 attributes, with 286 instances total.

3.2 Using Train and Test Sets for Anomaly Detection

In this section we describe our methodology in producing our subsets for anomaly detection. While describing our methodology of subset generation, we will use the examaple of weather-numerics dataset, which has 2 classes. However, this can be extended to an n-many class case easily. Class labels of the two classes in weather-numerics dataset are *"YES"* and *"NO"*.

While generating subsets, we first choose an anomaly class, say class *NO*. We select all the instances of class *NO* from the dataset and populate it to 1000 instances (100 instances will be given to the model at each era). This will be the so called *anomaly-test-set*. The remaining instances, i.e. *YES* class instances will be used to generate 90% of the *training set* or so called *normality-test-set*. The remaining 10% will consist of random instances simulated from anomaly class. The final training set will consist of 1000 instances (900 normals and 100 anomalies). Of course in the opposite case, we could have chosen the anomaly class as the *YES* class, in which case *anomaly-test-set* would be the populated *YES* class and so on.

In the previous paragraph we have described how we can generate the *normality-test-set*, *anomaly-test-set* for 2 class case. However, as can be seen the procedure is easy to generalize for an n-class case. In an n-class case, one of the

classes at each turn would be selected as the abnormal class to generate *anomaly-test-set* and the rest will be used to generate the *normality-test-set*.

3.3 Greedy Agglomerative Clustering Simulation Engine

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. 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.

A complete GAC tree also allows for data simulation. By mapping a dataset into a GAC tree structure we can query the leaf nodes in the tree to produce an arbitrary amount of new nodes which are a variable distance between a leaf and it's immediate parent. Data created this way will always lie close to a parent/child pair and will thus be similar the original datasets.

3.4 Experiments

Depending on which class of the dataset we choose as the abnormal class, we have 2 different treatments in our experimentals settings. In each treatment, we aim our anomaly dedection model to be able to identify both the abnormal and the normal cases.

In the first treatment, we choose one of the classes within a dataset as the normal class (say "YES" class) and one other class as the abnormal class (say "NO" class). The trianing has 1000 instances, whose members are simulated from *YES* class (90%) and from *NO* class . For the simulated train set, we have 2 test sets: *Normality-test-set*, *anomaly-test-set*. Both test sets are of **10 eras * 100 instances = 1000 instances**. We run our model on both test sets. In anomaly detection treatment, since our training set has only a 10% of its instances belonging to class *NO*, in the ideal case we expect all 100 instances of *anomaly-test-set* in era_i to be identified as abnormal cases by our model. Similar to *anomaly-test-set* instances being identified as anomalies, in the ideal case we expect all the *normality-test-set* instances to be identified as normal instances.

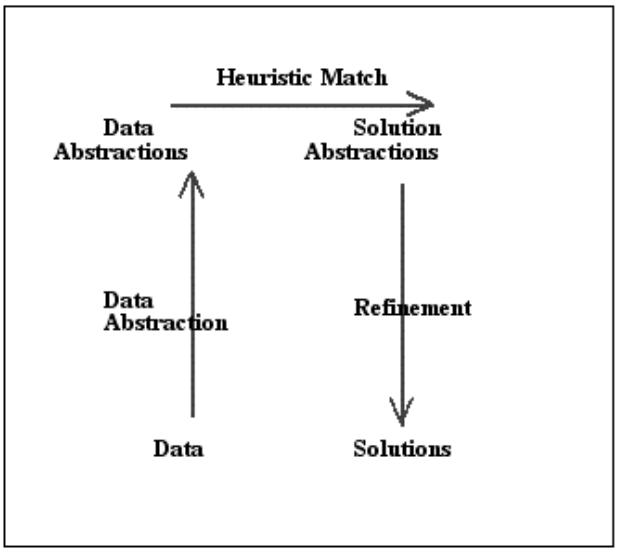
4. OUR MODEL

4.1 Building up the model

While building our model, we will use the three step procedure that was described in Section 2.1. In the following subsections we will describe these steps.

4.1.1 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". The template for heuristic classification is given in Figure 1.



Clancey's Heuristic Classification

Figure 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 likelihood-based two step procedure. Then hypothesis coming from the heuristic match is evaluated and the solution is reached.

4.1.2 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 them to be provided 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 dataset. Therefore, we will use a ***"NORMALIZE"*** method from the toolbox first. We also bin the numeric attributes into ten different bins and for that purpose we use the ***"DISCRETIZE"*** from the toolbox. Furthermore, since we use GAC tree to simulate our test and training sets, we will need a ***"GAC"*** method where we also 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. Our test instances go into the model one era at a time, which requires us to pick up the ***"ERAS"*** method from the toolbox as well.

4.1.3 Insert domain specific heuristics

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.

4.2 Executing the Model

Our model processes incoming new data in terms of separate eras, therefore we are able to process endless stream of data by letting the model work on it one era at a time. In our research each era contains 100 test instances, but one can change this number according to the performance and memory needs of a particular problem.

When the model starts processing an era, instances within an era are treated one at a time, i.e. model decides whether an instance is anomalous or normal one at a time. For learning what is abnormal and what is normal, the model relies on past data. For our case, we use 1000 instances in training set, in other words initially our past experience is made up of 1000 samples (1000 instances are generated by the GAC simulator).

The datasets we use in our research have both numeric and non-numeric attributes. We want to treat each attribute in a similar fashion, therefore we discretize the numeric attributes before conducting any computation on them. Furthermore, before using the data in our model, we normalize the test and training sets to remove the bias that would otherwise come from the range of different numbers. Once the data is normalized, the heuristic match mechanism will start to work. Our match mechanism or so called model is a likelihood based model that consists of a rejection/acception step. In the rejection/acception step, we decide for a test instance whether to accept it as a normal case or whether to reject it as an abnormal case.

The rejection step of the model aims both numeric and non-numeric features. In this step, we use a likelihood calculation. We first calculate the likelihood probabilities of all the instances in the training set. Then we take the minimum and the maximum probabilities from the training distribution. The minimum and the maximum probabilities give us an idea of probability range on which the so called normal instances are distributed. When we calculate the same likelihood probability for the test set, we expect its probability value to fall between the maximum and the minimum probability values of the normal instances. If the test instance has a probability value outside this range, then we reject it as an anomaly. On the other hand, if the likelihood probability of the test instance is in this range, then we accept it and call it a normal case.

During training, we produce a frequency table of class counts. This table contains attribute/range/class values for every instance in the training set which appear as a combination with a count. For example, [sex, male, pregnant] = 0. Using this table we calculate the likelihood of an incoming instance by determining the product of each attributes' probability in the instance.

$$product(\frac{frequency[attr, range, "seen"]}{frequency["seen"]}) \quad (1)$$

As stated previosly, we assign a probability of 1 to a numeric attribute which is within the minimum/maximum range of that attribute and a probability of 0 to a numeric attribute which isn't within the range.

Remember that we have a tuning mechanism in that model. We have minimum and maximum values that decide on whether an instance is to be labeled as an anomaly or not. We can play with these minimum and maximum indicators manually. If we increase the maximum value and/or decrease the minimum value in the model, then we enlarge the normality zone and let more instances be labeled as normal. However, if we do the opposite, i.e. decrease the maximum and increase the minimum indicator (thereby increasing the constraint to be a normal case) we will be more strict while selecting a normal case and therefore less instances will be labeled as normal.

Basically our model works in 3 steps:

1. Take one era of 100 test instances, let's say era_i
2. Label each instance in era_i as normal or abnormal
3. Correct anomalies in era_i, then check via a statistical test whether the correction worked or not.

The model whose specifications have been verbally given above is a generic likelihood-based anomaly detector, which is designed to work particularly for streaming data. For more detail, we provide the pseudocode in Figure 2.

	Normal		Anomalous	
Dataset	PD	PF	PD	PF
Breast Cancer	1	0.55	1	0.08
Clevaland Heart Disease	1	0.82	1	0
Cloud	1	0.16	1	0.61
CPU	1	0.9	1	0
Credit Rating	1	1	1	0
Ionosphere	0	1	1	0.01
KR-VS-KP	1	0.9	1	0
Mushroom	1	0.99	1	0
Splice	1	1	1	0
Contact lense	1	0.17	1	0

Figure 3: PD and PF results for 5% cut in low likelihood values. In every treatment we have 2 different experiments: Anomaly and normality. In 9 of 10 datasets our model attains very high PD rates (1). Furthermore, for the detection of anomalies, PF rates are also very low.However, the PF rates for normal instances are usually very high. In only 2 (contact lense and cloud) out of 10 datasets we have very low pf rates. Therefore, our model is quite successful in terms of detecting anomalies, yet suffers from high PF rates in normal case identification.

5. RESULTS

In this section we present the results of our anomaly detector. The summary of our results for all of the 10 datasets are provided in Figure 3 and Figure 4. After building the likelihood table for the instances in our train set, we prune away some of the instances depending on their likelihood values. For example, the difference between results in Figure 3 and Figure 4 is that in Figure 3 we prune train instances that

```
for all era_i ∈ testSet do
  for i = 1 to size(era_i) do
    testInstance ← ithInstance(era_i)
    trainingSet ← discretize(trainingSet)
    trainingSet ← normalize(trainingSet)
    minProb ← minimum(likelihood(allcolumns))
    maxProb ← maximum(likelihood(allcolumns))
    probTestInstance ← likelihood(testInstance)
    test1_min ← isBetween(probTestInstance, minProb, maxProb)
    test1_max ← isBetween(probTestInstance, minProb, maxProb)
    if isTrue(test1_min, test1_min) then
      testInstance ← normal
    else
      testInstance ← anomaly
    end if
  end for
  correctedInstances ← correct(anomalies)
  if Wilcoxon(correctedInstances, trainingSet) says they are the same then
    correctionSuccessful ← 1
  else
    correctionSuccessful ← 0
  end if
end for
```

Figure 2: Pseudocode for generic anomaly detector. Test-set can be a normal or an anomaly test set, i.e. it can consist of all abnormal test instances or all normal test instances. The anomalous instances are corrected and at the end of each era correction is controlled with Wilcoxon test to see whether it was successful or not.

	Normal		Anomalous	
Dataset	PD	PF	PD	PF
Breast Cancer	1	0.71	1	0
Clevaland Heart Disease	1	0.97	1	0
Cloud	1	0.67	1	0
CPU	1	0.99	1	0
Credit Rating	1	1	1	0
Ionosphere	0	1	1	0
KR-VS-KP	1	0.95	1	0
Mushroom	1	0.98	1	0
Splice	1	1	1	0
Contact lense	1	0.25	1	0

Figure 4: PD and PF results for 10% cut in low likelihood values. Getting rid of more difficult instances (low likelihood valued instances) improves the anomaly detection, i.e. we have PD's of 1 and PF's of 0. However, it increases the PF rates for normal instance identification.

have the lowest 5% likelihood values and in Figure 4 we do the same for the lowest 10%.

As we can see in Figures 3 and 4, our model is able to detect both the normal cases and the abnormal cases with very high pd (probability of detection) rates. Furthermore, for the anomalies our pf (false alarm rates) are extremely low except the cloud dataset. Although we also have very high pd rates in normal cases, our pf rates are high as well. Therefore, we can say that our model is quite successful in terms of detecting anomalies (high pd and low pf values). However, when the model is used for detecting normal cases, our false alarm rates are usually very high.

The actual difference between Figure 3 and Figure 4 is that 10% cut of low likelihood train instances helps our model to learn the normality in a better way. We can see the effect of getting rid of difficult-to-learn instances in the pf rates of Figure 4. As we can see the pf rates which had non-zero values in Figure 3 have all dropped down to zero in Figure 4. However, the drop in pf rates of anomaly detection comes with its own cost, slightly higher pf rates in normal instance detection.

6. ANOMALY CORRECTION

Whenever we reject anomalies, we provide user the option to correct the anomalies. We have conducted anomaly correction on all the datasets for the rejected anomalies and we were able to cure the anomalies. After anomaly correction, all the rejected instances were able to be accepted as normal cases. Therefore, apart form identifying anomalies, our model also provides means to fix them. In Figure 5 are a couple of examples from breast-cancer dataset.

During the processing of each era, the model goes through an sequence of steps to correct anomalies and check if correction really worked. The sequence of steps as well as what they mean and how they are employed in our model is as follows:

1. Monitoring: To observe each test instance at a time in our current era. In each era, we monitor the likelihood value of each test instance.
2. Diagnosing: To identify anything we think is out of line. Instances with either too low or too high likelihoods are identified to be taken care of.
3. Predicting: What's wrong with the diagnosed instance? We use contrast set learner to determine what is wrong with the diagnosed instance.
4. Controlling: Where we go from here? Are we going to fix it or not? We figure out how -if possible- we can correct the fault with the diagnosed instance.
5. Planning: Propose a change. If we can propose a solution of correcting the fault in controlling phase, then

Anomalies
1- (60-69 GE40 20-24 0-2 NO 2 LEFT LEFT_UP NO NO-RECURRENCE-EVENTS)
2- (40-49 PREMENO 35-39 0-2 NO 2 RIGHT RIGHT_UP NO NO-RECURRENCE-EVENTS)
3- (50-59 LT40 20-24 3-5 NO 2 LEFT LEFT_LOW NO NO-RECURRENCE-EVENTS)
Corrections
1'- (60-69 GE40 20-24 0-2 NO 2 LEFT LEFT_UP NO **RECURRENCE-EVENTS**)
2'- (**50-59** PREMENO **50-54 9-11 YES 3** RIGHT LEFT_UP NO **RECURRENCE-EVENTS**)
3'- (50-59 **GE40** 20-24 3-5 NO 2 LEFT **LEFT_UP** NO **RECURRENCE-EVENTS**)

Figure 5: Sample anomalies and their corrected versions. Corrected version of anomaly 1 is shown with 1' and so on. The corrected features are shown with bold face.

we plan a change that would result in the proposed solution.

6. Repair: Apply the change. the planned change is applied and the instance is corrected

7. Success: Were we successful with the previous steps? This step is executed at the end of each era and not for each test instance. At the end of each era, we control if we fixed all the diagnosed instances. For that purpose, we apply a Wilcoxon Test on the likelihood values of train as well as on those of corrected era-instances and if Wilcoxon Test says that they are statistically the same, then we say that this era was successfully repaired.

7. CONCLUSION

In this research we identified a generic procedure as well as a toolkit to aid anomaly detection and repair procedure. Furthermore, by following the proposed generic procedure we developed a sample expert system for anomaly detection in classification datasets with 2 or more classes. According to our results, our model is extremely successful in terms of diagnosing and fixing anomalies. For anomalies we have very high pd and very low pf rates. The model's performance in terms of identifying the normal cases is questionnable. Model can attain very high pd rates for normal instance identification. However, at the same time it has very high pf rates. Therefore, our model can be used to identify anomalies in classification datasets, but if it is to be used on normal instance identification it will come with the cost of high pf rates.

The model is also capable of repairing the detected anomalies. Our experiments have shown that we are able to propose a solution (fix) all the anomalous instances. We also control whether the success of our fixing procedure with a statistical test. Statistical test has also shown that we were able to correct whole eras as well.

A further contribution we made in this research is to use GAC simulating engine. We used GAC simulating engine to generate/simulate similar but unique datasets out of a single classification dataset.

8. 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] R. Reboh. Extracting useful advice from conflicting expertise. In *IJCAI'83: Proceedings of the Eighth international joint conference on Artificial intelligence*, pages 145–150, San Francisco, CA, USA, 1983. Morgan Kaufmann Publishers Inc.

[6] 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.

[7] E. H. Shortliffe. *Mycin: a rule-based computer program for advising physicians regarding antimicrobial therapy selection.* PhD thesis, Stanford, CA, USA, 1975.

[8] A. O. Tim Menzies, David Allen. Bayesian anomaly detection (bad v0.1). 2006.

[9] S. M. Weiss, C. A. Kulikowski, and A. Safir. A model-based consultation system for the long-term management of glaucoma. In *IJCAI'77: Proceedings of the 5th international joint conference on Artificial intelligence*, pages 826–832, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc.

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

[11] W.-K. Wong, A. Moore, G. Cooper, and M. Wagner. What's strange about recent events (wsare): An algorithm for the early detection of disease outbreaks. *J. Mach. Learn. Res.*, 6:1961–1998, 2005.