Introduction

The problem with anomaly detection\textsuperscript{1} using machine learning methods is that they usually give results that are hard to explain, even if they are correct. Customers want to catch anomalies as they think what anomaly is, not what an autoencoder or a clustering finds\textsuperscript{2} as anomaly. What I have found in my experience is that instance-based learning methods are much more useful and explainable in this domain than any other methods and they are more compatible and acceptable by human's mindset. In a nutshell, instance-based learning tries to mimic the way human being thinks, the most famous method in this learning paradigm, we all are familiar with, is the k-nearest algorithm. In this white paper we want to design an algorithm to find anomalies using instance-based learning methods\textsuperscript{3}.

The problem

It is simple; we are monitoring and observing a system and want to know how much the latest observation is off from the typical behavior of the system? We have nothing but the observation. No labels or supervising, all we know is that in long-term the system is more in its normal behavior than abnormal.

\begin{equation}
  o_1, o_2, o_3, \ldots, o_t
\end{equation}

(1) This is the result of our system observation

In (1) we do not care how many dimensions any of our observation results have, we will see that it only changes a distance function we need to provide. Defining a distance function is not strange, it is the very first thing any biological systems or machines need to retrieve knowledge from data.

Time of observation and the algorithm idea

Since our model does not use any labeled data, learning means adding new observations to the model and removing the old ones. So, here, time is so much valuable even if our problem is not time-series. Let’s have two examples to show how we should deal with time in this learning methods.

Example 1: Consider we want to find out how much a credit card transaction is normal (or abnormal). We usually ask ourselves the following questions and find the answer:

1. Does this customer has had such a transaction before?

\textsuperscript{1} An anomaly can be defined as any observation from a system deviating from the expected result.

\textsuperscript{2} This is something else that we can talk about it somewhere else.

\textsuperscript{3} This is not a right place to talk about, but my experience also shows that we can build high-performance pattern recognition algorithms using instance-based learning method too.
2. If no, the transaction is abnormal.
3. If yes, now the question is when? How many times?
4. Now based on the number of the same observations, their similarities with the new one and the time of the observations we do some rough calculation in our mind and say:
   a. OK, we have seen enough of such transaction, so it is normal.
   b. No, we just have had a few similar transactions many years ago, so it is abnormal.
5. We will discuss the detail of finding similar transaction later, but for credit card transactions, we might be just interested in checking not all part of the available past information. For example, if the given transaction has happened on a Sunday 10:30 pm, we think just for any Sunday nights’ transactions nothing else, unless we want to apply more approximation, in this case we might consider Saturday nights’ transactions too, or Sunday’s anytime.

As you see it mainly works like k-nearest (or nearest neighbor), but instead of looking for similar ones around, in Euclidean space; we just look to the past and try to find similar transactions. Here unlike k-nearest we also deal with fuzzy similarity values, and the amount of closeness or apartness of every observation depends on observation time, not the physical distance.

Example 2: Suppose we are monitoring or observing download volume of network traffic. The question again is that how much the current download is normal? Precisely like the credit card example, we just think about it as "Do we usually observe the same amount of download volume every Sunday night?" We do not care that much if it is 10:30 pm or 10:00 pm, We just use our memory of the past and do a simple comparison between current observation and stored Sunday nights traffic patterns to find out if the current observation is OK.

**Aggregation, Coarse Coding and Feature Extraction**

Credit card transactions are so diverse in amount, type, place, etc. so, we need to choose proper coarse units to keep them in memory, otherwise high precision values doesn’t give us anything good. For network traffic, the data is usually too much and noisy, since we cannot process data every second, so again we might just need a window to aggregate data. Now the problem is how we can aggregate or coarse code these data, for network traffic, considering observing just one single parameter, we can calculate the average of the observation and convert it to a point like \((\text{dayOfWeek}, \text{timeOfDay}, \text{trafficValue})\). Note that if we choose coarse time unit of 15 minutes, \(\text{timeOfDay}\) is from 0 to 95 and for credit card transactions it might be like \((\text{dayOfWeek}, \text{timeOfDay}, \text{weekOfMonth}, \text{amount}, \text{place}, \text{category})\). Here, \(\text{timeOfDay}\) could be just 0,1,2,3 representing midnight, morning, afternoon and night, and \(\text{weekOfMonth}\) shows in which week of the month the transaction has happened. Since it is essential to learn the behavior of the customers; we have to know when they usually do their high or low amount transactions, some people pay their rents in the first week of the month some in the middle, etc. this is why we also have \(\text{weekOfMonth}\) here. So, the shown observation in (1) will become something like what we have in (2):

\[
O_1, O_2, O_3, ..., O_t
\]

(2) This is the aggregated and coarse coded data

---

4 Or sparse in any dimensions
5 This is the human’s power, instead of fine measuring of one or two feature, uses coarse measuring of more features. Unfortunately, we tend to use fine measurement and calculation just because computers give us such a power.

\[
24 \times \left( \frac{24}{12} \right) - 1 = 95
\]
Basically, what we do in the first step is aggregating and/or coarse coding the data and storing them somewhere. These stored objects have denser information than the original observations, and when it comes to finding anomalies, we are dealing with much fewer data points. Do not forget by doing these transitional processes; we already have saved a lot of time for the times we need to calculate the anomalies.

**Similarity and Distance function**

If you want to run any clustering on a dataset to extract some information you need a distance function. The problem with available embedded distance functions in most of ML algorithms is that they are so general and if you use them you usually miss so much of information. I personally, think it is wrong not to use the domain/business knowledge we have and rely just on some simple Euclidian or any other mathematical distance functions. So, we have to define a function that returns the similarity or distance between two given observations. Distance functions should have some fundamental properties called “pseudometric” as below:

- \( \text{distance}(O_1, O_2) > 0 \)
- \( \text{distance}(O_1, O_1) = 0 \)
- \( \text{distance}(O_1, O_2) = \text{distance}(O_2, O_1) \)
- \( \text{distance}(O_1, O_2) + \text{distance}(O_2, O_3) \geq \text{distance}(O_1, O_3) \)

So, if you define a function that satisfies the above conditions and accepts two different observations and returns a value, you can use it in this algorithm or any clustering methods. Now if our distance function returns a normalized value, the similarity can be considered as (3):

\[
\text{similarity}(O_1, O_2) = 1 - \text{distance}(O_1, O_2)
\]

(3) The relationship between normalized distance function and similarity

**Threshold vs Score**

The nature of the credit card transactions and network traffic are different. For credit card, we might accept it is normal, if our past observations show, the customer has had enough of the same transactions, say ten times. However, for network traffic, it not like that, we might need to compare all available past observations with the new one and see how many times it is like the past observations and how many times it is not. These two numbers, similar observation count, and dissimilar observation count are calculated based on an already given threshold. Now, if the similar observation count is higher than dissimilar observation count, the current observation is normal. Otherwise, it is not.

**The anomaly detection process**

Here is the summary up to now, we have series of observations shown in (1), the very first process converts them to aggregated and/or coarse coded data objects shown in (2). Now the question is what is the process that defines how much the observation \( O_t \) is normal? Here is a sample version of the algorithm, step by step:

---

*We can do better and build more informative and dense information at this step too. These objects are in fact the grasped idea of every small window of the past. t*
1. Find all the past observations like $O_t$ having same coarse time of $O_t$. For credit card it means all the past experiments that happened at the same coarse time $(\text{dayOfWeek}, \text{timeOfDay}, \text{weekOfMonth})$, and for the network traffic it means those happened at $(\text{dayOfWeek}, \text{timeOfDay})$. Let's call this sub set of observations as $\text{SimilarTimeObservationSet}(O_t)$.

2. Not all observations in $\text{SimilarTimeObservationSet}(O_t)$ have same effect or value for scoring or calculating how much normal or abnormal is $O_t$. Definitely, those have observed two years ago don't have enough value as those have happened month ago. So, we have to calculate this factor which shows how much we have to learn from any of these observations, let's call this function $\text{learningFactor}(O_i)$.

3. For all or as much as required, those observations that have above the zero $\text{learningFactor}(O_i)$ we need to calculate $\text{similarity}(O_t, O_i)$ or $\text{distance}(O_t, O_i)$. 

4. Now with the given $\text{minimumSimilarityThreshold}$ we can say how many of these observations in $\text{SimilarTimeObservationSet}(O_t)$ is similar to $O_t$, remember we assumed that the $\text{similarity}(O_t, O_i)$ is a normalized number between 0 and 1. So, for example, with $\text{minimumSimilarityThreshold}=0.2$ any $O_t$ with similarity less than 0.2 will be considered as not similar.

5. The rest is just calculating the sum of the $\text{learningFactor}(O_i)$ for two groups, similar ones and dissimilar ones.

Study the provided listing (3), which might give you a better idea of how the algorithm works. This algorithm is for domains like network, not credit card transactions. For credit card transactions we might exit the main loop as soon as we get enough score and know the transaction is normal.

```
similarScore = 0
dissimilarScore = 0

for all $O_i$ in $\text{PastObservationSet}$ do {
    if time of coarse observation of $O_i$ is similar to $O_t$ {
        if $\text{learningFactor}(O_i) > 0$ {
            if $\text{similarity}(O_t, O_i) > \text{minimumSimilarityThreshold}$ {
                similarScore += $\text{learningFactor}(O_i)$  // or similarScore += $\text{learningFactor}(O_i) \cdot \text{similarity}(O_t, O_i)$
            } else {
                dissimilarScore += $\text{learningFactor}(O_i)$  // or dissimilarScore += $\text{learningFactor}(O_i) \cdot \text{dissimilarity}(O_t, O_i)$
            }
        }
    }
}

anomalyScore = 0
if similarScore == 0 {
    anomalyScore = 1
} else if dissimilarScore == 0 {
    anomalyScore = 0
} else {
    anomalyScore = similarScore / (similarScore + dissimilarScore)
}
```

(3) The idea of anomaly detection algorithm

---

8 For credit card transaction you just need to check if such a transaction has happened before for a specific number of times or not, so you might not be required to loop over all available time similar observations.
Note that in (3) the only undefined terms or processes are distance and learning-factor functions which we will talk about them right away.

**Distance Function**

Our distance function does not have to do anything with the time because we consider the effect of time by applying learning-factor. Distance function should compare given observations and return a number between 0 to 1 which shows how much the two given observations are apart from each other. A sample distance function for single value could be like what we have in (4).

\[
\text{distance}(O_1, O_2) = \frac{|O_1.\text{value} - O_2.\text{value}|}{\text{Max}(O_1.\text{value}, O_2.\text{value})}
\]

(4) Sample distance function for observations with single numeric value

For observations with multiple dimensions like credit card transactions, you have to define a particular distance function, based on the domain knowledge you have. For example, a credit card observation might have \((\text{amount, place, category})\) as well as time. So, the distance pseudometric function should get a pair of \((\text{amount, place, category})\) and return a number between 0 and 1. Since it highly depends on domain knowledge, we have many options; here is one we suggest in (5). It says the overall distance is more dependent on amount than the other two parameters.

\[
\begin{align*}
\text{categoryDistance}(O_1, O_2) &= \begin{cases} 
0, & O_1.\text{category} == O_2.\text{category} \\
1, & O_1.\text{category} <> O_2.\text{category} 
\end{cases} \\
\text{placeDistance}(O_1, O_2) &= \min \left( \begin{array}{c} 
0, \\
0.5, \\
1, 
\end{array} \right) \begin{align*}
&O_1.\text{place.city} == O_2.\text{place.city} \\
&O_1.\text{place.country} == O_2.\text{place.country} \\
&O_1.\text{place.country} <> O_2.\text{place.country} 
\end{align*} \\
\text{amountDistance}(O_1, O_2) &= \frac{|O_1.\text{amount} - O_2.\text{amount}|}{\text{Max}(O_1.\text{amount}, O_2.\text{amount})} \\
\text{distance}(O_1, O_2) &= \frac{2}{3} \times \text{amountDistance}(O_1, O_2) + \frac{1}{6} \times (\text{categoryDistance}(O_1, O_2) + \text{placeDistance}(O_1, O_2))
\end{align*}
\]

(5) Sample distance function for multi-dimensional observations
Learning Factor

As we said before, learning-factor just depends on the time of the observation. Look at figure (6) it says if the observation is closer than two weeks or older than one year from now, we do not learn from it. This is one of the best strategies to learn from unknown data. You never have to hurry to learn new stuff and should always forget the old ones. Not from new ones, because in case they are abnormal behavior, if we learn them as they come, they might change our judgment fast. And not from very old ones, because the system’s behavior might have changed.

The problem with (6) is that almost no phenomenon in nature works like that, natural phenomena usually follow or change in a smooth and curved pattern. So, figure (7) or (8) might be a better form of defining the learning-factor function. Note that using different learning-factor functions is like the way different people remember the past.

---

9 Usually exponential patterns like saturation or s curve.
About the algorithm
The very first question comes to the mind using this method - comparing to ordinary ML methods - is that, why should we perform many calculations whenever we want to see if an observation window is an anomaly? The answer is we do not do too much process to find anomalies. We should not forget the repetitive, high cost and painful process of training an ordinary ML algorithm like NN or clustering, etc. Also, do not forget we do not compare current observation window with the entire available past data. We use two filters that narrow downs them, the learning-factor function and the similar time functions. So, for a user with say 8 credit card transaction per day - which is not normal - using the above models, you need to do just $\frac{8}{4} \times 54 \times 1 = 108$ comparisons\textsuperscript{10}, which is nothing\textsuperscript{11}.

The next question is how much this algorithm is correct? We cannot calculate the accuracy, or any other confusion matrix extracted indices unless we have access to labeled data. However, the reasoning model is precise, it is what we do on almost everyday decision makings, so we do not have problem proofing to the customer that how the machine has reached to such a conclusion, and my experience shows they always accept the results. Because first, they understand the algorithm, and second, they already have cooperated to define the distance functions.

Instance-based learning methods have many benefits, one of them is that you can build the model for every individual separately. You do not necessarily need to use all the individual's data together to build a generalized model which gives you a model for the overall behavior of individuals. It means, in credit card example, the algorithm learns the behavior of every customer not a general behavior for all the people or groups of them.

There is another significant benefit if we use this algorithm, for learning the behavior of different individuals, we only update the model when there is some new observation available for that particular individual, not for all. However, for batch updating algorithms, you always need to update model using the whole data, the reason is a single individual data usually is not enough to build a model using auto-encoder or clustering.

Moreover, we should not forget this algorithm starts learning as the first observation comes real-time, and day by day it gets a better idea of the behavior of the monitoring system.

Test results
I have tested this algorithm in different real-world applications and data sets, another benefit of this approach is that it works efficiently even with uncleaned and noisy data. Look at the shown data in figure (9), and you clearly see how messy the data is there, sometimes you do not have data at all, etc. However, if you let this algorithm learns the data after like six months, it starts giving you very correct and acceptable information. Because at that time it has like 25 days for every single day of the week and it is enough to see what the dominant behavioral pattern of each day is unless in these days - at that part of the day - you always observe abnormal data, which is against our only initial assumption. In this case, the algorithm considers that behavior as normal. Figure (10) shows some of the results after this training period. If you carefully check the results, you'll see the algorithm learns the behavior of the data and shows abnormalities perfectly well.

\textsuperscript{10} There are 54 weeks in a year, also, note that dayOfWeek for financial transactions could be just either working day or weekends.

\textsuperscript{11} Compare it with iterations over the massive amount of data you need to do to train a supervised or unsupervised algorithm.
(9) Some daily sample of real world messy data
(10) The anomaly detection result of the algorithm