Universiteit Utrecht

Applied Data Science
Natural Sciences
Utrecht University
3508 TC Utrecht, The Netherlands

Subject

Anomaly Detection Techniques on Relational Data as Quality Evaluation of a Dataset

Author: Emmanouil Iliakis

July 21, 2022

Author: **Emmanouil Iliakis**
Email: e.iliakis@students.uu.nl
Adviser: Professor Dr. Yannis Velegrakis,
Utrecht University

Abstract

An outlier is a point that deviates significantly from the pattern that has been formed from the majority of the data points. The presence of outliers can exacerbate statistical results which leads to misrepresented relationships between different data and faulty conclusions based on them. This is an urgent issue in a data driven world and people in the data sector are following harsh and tedious procedures to deal with that. In the present article, an half automated tool for outlier detection returning a single score for a structured dataset is proposed with minimal human intervention. This tool can either be used in a python environment or directly in a python script.

Key Words: Outlier detection tool, Autoencoder, ngrams, Isolation Forest, Lightweight Online detection of anomalies

Contents

Chapter 1

Introduction

In the past 20 years, there has been a prodigious increase in available data, particularly in the big data section as measured by bibliometric analysis by Raban and Gordon (2020) , which in turn increases the already high demand for data scientists to make use of this data to benefit whoever is in need. One way to moderate the need for data scientists is to aid the existing ones to work more efficiently. Data scientists spend about 60% of their working time on cleaning and organizing datasets and 19% of it on collecting datasets according to Maqsood et al. (2017). Thus, it is safe to assume that by automating the process of evaluating a dataset in terms of their anomaly will benefit data scientists as they will not waste time on organising datasets which would eventually proven to be too anomalous for the intended use such as prediction making. Furthermore, anomaly detection is crucial to increase cyber security as it can be used to detect cyber intrusions that are likely to cause significant damage to various objects ranging from databases Sallam et al. (2016) even to the power grid itself Sallam et al. (2016).

The purpose of this paper to propose a tool which can assign an anomaly score to any given structured dataset without supervision from user apart from importing some initial parameters regarding the nature of the dataset as well as the preferred method of detecting outliers. The only needed inputs are the path of the dataset of interest, the method that will be used for outlier detection, a threshold which determines the cut-off point between an inlier and an outlier and a function to aggregate the individual scores of rows or attributes to calculate the overall score of the dataset. Parameters for the nature of the dataset are asked to determine further parameters needed for the method of detection such as number of epochs, number of layers and neurons per layer, activation function in hidden layer, batch size and number of estimators. Furthermore, the available methods for the outlier detection are the Ngrams, the Autoencoder(AE), the Isolation Forest(IF) and the Lightweight online detection of anomalies(LODA). The first one is used for string type data while the other three algorithms are used for numeric type of data.

However, before delving deeper into the tool, it would be wise to elaborate on what is considered an outlier. An outlier is a value or set of values which deviates significantly from the pattern formed by the majority of the values. As explained in detail by D. and Sasidhar Babu (2016), there are 3 types of outliers, global, contextual and collective outliers. Global are the outliers with an extremely high or low value in comparison to the rest of the values, contextual outlier is a data point whose value is in accordance with the rest points, yet it is in the wrong

context. For instance, a temperature value of 30 degrees Celsius is not extreme for the summer, however it is an contextual outlier if such a temperature is noticed in the winter. Last but not least, a collective outlier is a set of points which on their own have 'normal' values, though if observed collectively they deviate from the pattern. For example, even though it is normal for one of your neighbors to be moving out on a day as people do this quite often, it is considered a collective outlier if the whole neighborhood moves out on the same exact day. There are numerous types of anomalies as mentioned by Foorthuis (2021), but in this article mostly addressed with 3 main types of outliers and some types of anomalies such as the multidimensional numerical anomaly, where multiple quantitative attributes participate in an anomaly with each of them having values that comply with the general pattern, though when combined together those attributes create a data point that deviates significantly from the combined pattern. For example, if you let 'acceptable' values for height and weight of the general population be 160cm to 190cm and 60kg to 90kg respectively and a data point has values of 185cm and 65kg, it is an outlier. Although, those values are in the range of 'acceptable' values, this data point deviates significantly from the pattern that is formed because of the rest of the data points, thus it should be labelled as an outlier.

This article proceeds as follows. Chapter 2 discusses the work that has already been done, chapter 3 explains the fundamental characteristics of the outlier detection methods as well as 3 different approaches to quantify and measure the anomaly of a dataset, chapter 4 discusses the performance of the tool in two different datasets, chapter 5 concludes the key points of this article and proposes some steps that can examined in future research and chapter 6 discuss the results that have been found and the reasons behind the performance of the tool.

Chapter 2

Related work

Ngrams is a contiguous sequence of n items from a given sample and can be utilized to identify outliers in a dataset. Regarding the this method numerous variations exist such as constructing a model of normality and label as an outlier what does not coincides with it. However, as mentioned by Li et al. (2003) these methods suffer from the limitation of high false positive rate which was addressed in the same paper by identifying the critical parameters and optimizing them to reduce the false positive rate. Later on, classification of the data points is done which is essentially a discrimination between inliers and outliers N-grams is learned. Zak et al. (2017) provided evidence that byte n-grams can learn from the code sections of a binary as well as made a comparison between assembly and byte n-grams with byte n-grams demonstrating higher discriminative ability. The outlier detection method works with a moving window of n characters length and slides on the values of an attribute counting the times each gram occurs. In case the input is a pair or triplet of attributes, the values of the attributes are concatenated, and the algorithm search the Ngrams that have some character from the one value and some of the other value. This way it is ensured, that the information of the combination of the attributes is taken into consideration. After the results are gathered, the grams with unexpected low frequency are labelled as the outliers as described in algorithm 1. This type of algorithm due to its adaptability can be used on any data type whether it is float, integer or string as all of those types can be casted easily into a string type. Ngrams is useful in attributes such as telephone numbers, where specific values are expected and can not be treated as normal integer values. For example, in telephone numbers there is always a plus sign followed by a 2 or 3 digit code indicating the country this number was purchased. In this kind of attributes the digit code have a finite number of combinations which will occur numerous times in a big enough dataset, hence if an ngram occurs way less times than the rest of them then it's probably a mistake and can be labelled as an outlier.

The literature acknowledges various ways to distinguish between different manifestations of anomalies. Foorthuis (2021) makes a distinction between attributes to quantitative, qualitative and mixed attributes as well as univariate and multivariate cardinality of relationship. Sakurada and Yairi (2014) proposes the use of non linear dimensionality reduction with autoencoders as a better performance tool for anomaly detection than linear and kernel PCA and is tested it on both artificial and real data presenting that autoencoders understand the normal state

Algorithm 1 Ngrams training and classification

Input: data samples $\{x_i\}_{i=1}^{n}$, threshold th
Output: A list of outliers
initialize outliers;
initialize Ngrams;
for $j \leftarrow 1$ **to** n **do**
add ngram to Ngrams;
end for
k = length(Ngrams);
$\hat{f} = \frac{1}{k}\sum(f_i)_{i=1}^{k}$;
for $j \leftarrow 1$ **to** k **do**
if $f_j < \hat{f}$ **then**
add $ngram_i$ **to** outliers;
end if
end for
return outliers;

effectively and respond to anomalous input in a different way. They shown that autoencoders can decrease work load by omitting complex computaions that other algorithms need. Later on, Zhou and Paffenroth (2017) extended anomaly detection with autoencoders further by achieving elimination of outliers and noise without access to any clean training data as often it is not available. In order to do that different methods for handling missing data are tested on different outlier detection algorithms such as LODA and IF.

The Autoencoder algorithm is a deep fully connected neural network whose fundamental notion is predicting its own input. In more detail, this network's input layer consists of as many neurons as the attributes of the dataset and the input values are the values of the attributes of one of the row of the dataset (data point). Then, in the hidden layers, neurons are fewer consecutively in each layer up to a certain number of neurons which is called bottleneck. The output of the bottleneck provides a reduced representation of the data point. What happens through this process is that information of the data point is compressed up to a certain point which is determined by the bottleneck and then the information is decompressed through a symmetric but opposite process. In this symmetric and opposite process, layers have progressively more neurons until the output layer has the same number of neurons as the input layer. This process can be thought of as decompressing the information progressively in each layer. A deep network similar to an autoencoder network can be seen in figure 2.1, where x_i is the input values of the i attributes and the x_i' is the predicted by the network value. Then a loss function, such as the mean squared error, is used to measure the difference between input and output and after the proper training value of the loss function is minimized. The loss function indicates the anomaly score of the data point and the aggregation over all the data points is the anomaly score of the dataset. The loss function is expected to take low values in the majority of neurons, whereas for the outliers the difference will be higher as the network is trained to predict the input under the assumption that the percentage of the outliers is low, thus the predictions always follow the pattern of the normal values. If the difference between input and output exceeds the threshold then that data point is assigned as an outlier. The autoencoder is a straight-forward solution for outlier detection as it is usually used for dimensionality reduction

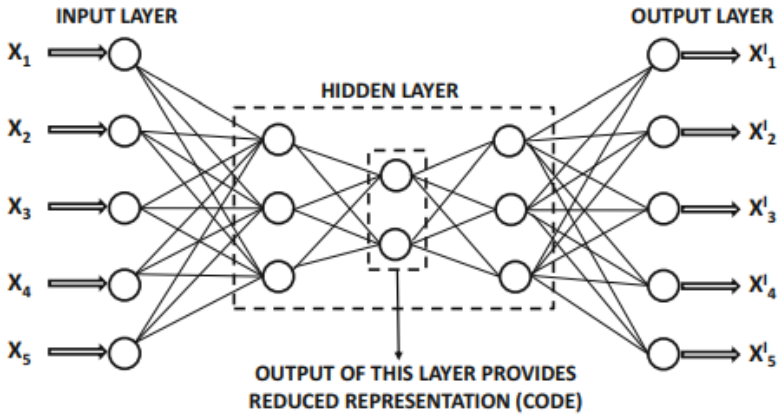

Figure. 2.1: The autoencoder architecture. The two neurons in the middle hidden layer is the bottleneck of the network.

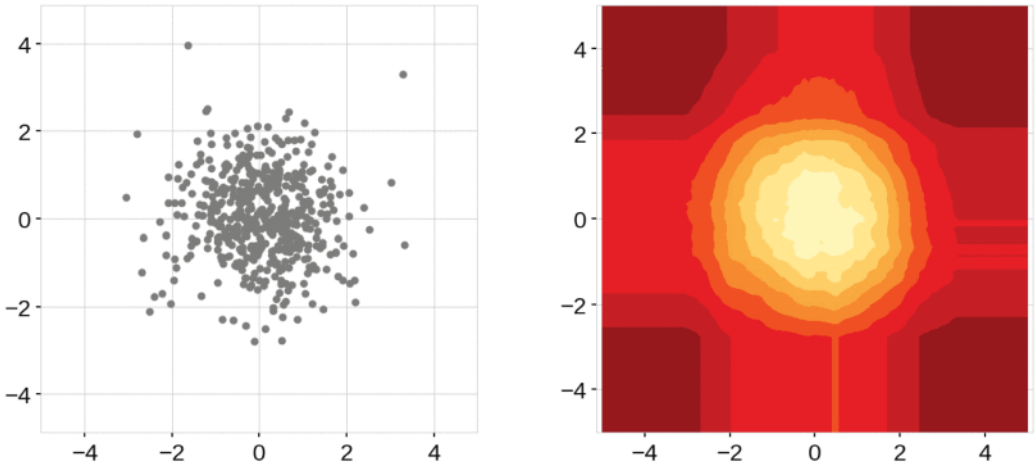

Figure. 2.2: Two dimensional normal distribution of data points with $(\mu_1, \sigma_1, \mu_2, \sigma_2) = (0, 2.5, 0, 2.5)$ (left plot), the anomaly score assigned to these points by the IF algorithm where lighter colors indicate lower anomaly score (right plot).

of multidimensional data because the number of neurons in the hidden layers are significantly fewer than the input which means that can be viewe as a reduced representation of the data as mentioned by Aggarwal (2016). The reason behind autoencoders yielding good results is that through the dimensionality reduction the information is compressed and patterns are learned while through the opposite process of decompression the learned patterns are tested to recreate the input.

Finally, the autoencoder suffers from two main limitations which are common for most of the outliers detection methods. Firstly, it can work only on numerical values and secondly if the percentage of the outliers is the dataset is high it can not yield good results. Unfortunately, these problems are inherent to the model and can't be dealt.

Another outlier detection method, that does not require a deep network to be trainied is the Isolation Forest. Nine years after the first sight of anomaly detection by isolation of outliers by Liu et al. (2008) an improved method was proposed by Xu et al. (2017), which is based on

the idea of selective integration with the precision as the criterion and the simulated annealing algorithm is used to select the tree with the highest abnormality detection. Meanwhile, the efficiency of the algorithm is increased as the excess detection precision is avoided with minor change in the Isolation Forest. As proposed by Liu et al. (2008) IF works fundamentally different to Autoencoder and Ngrams. The basic principle is that instead of measuring the difference between predicted and true value, it projects all the values of a pair/triplets of attributes into a feature space where each attribute is represented as one dimension. Then, in each iteration a random dimension is chosen and a split on it occurs at a random spot between the minimum and the maximum of the values in the respective dimension. This random split in a random dimensions reoccurs until in each hyper cuboid at most one point is contained, thus every point is isolated from the rest of the points or the upper limit of iterations is reached. A binary tree is created where in each split the random value is the threshold and if the data points exceed this threshold, they are assigned to the right of the leaf, otherwise to the left of it. In the end, the depth that each point has reached is measured (number of needed splits to isolate the point is counted) and the lower the depth in the tree (number of splits), the more likely it is for the point to be an outlier. Then, the whole process is repeated multiple times so many binary trees are created. At this point, the training process is completed and each new point going through the this algorithm goes in each trained tree separately and based on the number of splits in all the trees an anomaly score is assigned to the data point. In the case, there is only one attribute to be examined for anomalies the first step of picking a random dimension in the feature space is omitted as it is an one dimensional space. This method suffers from a significant limitation which is the introduced bias by the method itself as mentioned by Hariri et al. (2021). This can be easier interpreted by an example. Let a normal 2 dimensional Gaussian distribution of data points and the anomaly score as given by the Isolation Forest algorithm as seen in 2.2. What stands out from the anomaly score plot is that there are four rectangles around the circle which have assigned lower anomaly score than their direct neighbors even though there are not data points at these areas. Hence, if a test point is in those rectangles it might be assigned as an inlier even though it would not be one. This kind of bias is inherent to the algorithm as each split at a random value is essentially a straight line at a given point and can be noticed even with more than one normal distribution with more biased rectangles being formed as seen in 2.3.

Algorithm 2 IF's training routine for k dimensions.

Input: data samples X,number of trees t
  initialize Forest;
  for $i \leftarrow 1$ **to** t **do**
    while points not isolated or steps $<$ limit **do**
      $dim \leftarrow random(k)$;
      $split \leftarrow min(dim) + random(range(dim))$;
    end while
    $iTree \leftarrow binary\ tree\ based\ on\ splits$;
    $Forest \leftarrow Forest \cup iTree$;
  end for
  return Forest;

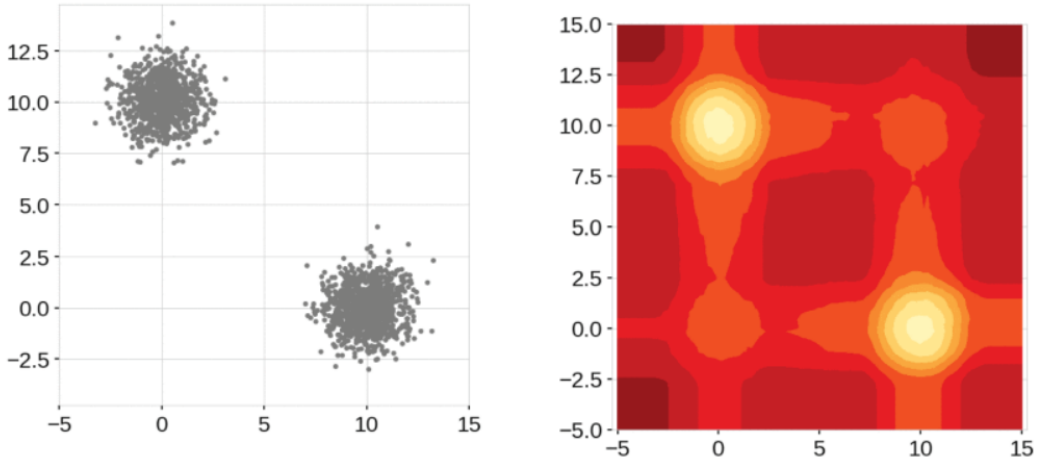

Figure. 2.3: Two two-dimensional normal distributions of data points with $(\mu_1, \sigma_1, \mu_2, \sigma_2)_1 = (0, 2.5, 10, 2.5)$ (top-left cluster) and $(\mu_3, \sigma_3, \mu_4, \sigma_4)_2 = (10, 2.5, 0, 2.5)$ (bottom-right cluster) (left plot), the anomaly score assigned to these points by the IF algorithm where lighter colors indicate lower anomaly score (right plot).

Zemicheal and Dietterich (2019) followed a different approach using LODA to address the issue of streaming weather data which may be faulty due to a dysfunction in the weather sensors and proposed some strategies to handle missing values in anomaly detection. Although, this is useful as missing values occur in most of the data sets, no proposals had been done to increase the performance of LODA in its core, which was done 2 years prior to that by Saarinen (2017), who, initially, handled special cases by adding a weight to different data points so all data points do not affect the result equally as well as testing the limits of the method when data is not pre-processed and proposed different techniques to set the threshold of determining if the anomaly score is high enough for the point to be labelled as an outlier. LODA is an ensemble of one-dimensional histograms which are weak learners, though a collection of them seem to yield satisfactory results. Briefly described this algorithm initially selects each data point of the dataset iteratively and selects a random subset of attributes which consists of the reverse square root of the number of attributes of the initial dataset, which according to Li (2007) can preserve the euclidean distances between points in the projected space and the input space. Then, based on the chosen attributes it creates/updates the histogram about the probability density of occurrence of values in each dimension separately as seen in algorithm 3. LODA determines if a new data point is an outlier or not depending on the loglikelihood of all the values of the data point co-occurring simultaneously according to the histograms that have been created in the training process as described in algorithm 4. The word `online` in the name of the method means that the model is trained continuously on new incoming data and adapts to incoming traffic further and further.

After the building of histograms is completed, for each incoming data point the negative loglikehood of the combination of values is measured, which means that the less likely a tuple is, the higher the anomaly score will be and if the score exceeds a threshold, the tuple is labelled as an outlier. The most obvious flaw of this method is that it works under the assumption that probability distributions on different attributes are independent, whereas that does

Algorithm 3 Loda's training (update) routine.

Input: data samples $\{x_i\}_{i=1}^{n}$
Output: histograms $\{h_1, ..., h_n\}$, projection vectors $\{w_i\}_{i=1}^{k}$ initialize projection vectors with $\left[d^{-\frac{1}{2}}\right]$ non zero elements; initialize histograms $\{h_1, ..., h_n\}_{i=1}^{k}$;
  for $j \leftarrow 1$ **to** n **do**
    for $i \leftarrow 1$ **to** k **do**
      $z_i = x_j^{\mathbf{T}} w_i$;
      update histogram h_i by z_i;
    end for
  end for
  return $\{h_1, ..., h_n\}_{i=1}^{k}$ and $\{w_i\}_{i=1}^{k}$;

Algorithm 4 Loda's classification routine on sample x.

Input: sample x, set of histograms $\{h_1, ..., h_n\}_{i=1}^{k}$ and projection vectors $\{w_i\}_{i=1}^{k}$;
Output: anomaly value f(x);
  for $i \leftarrow 1$ **to** k **do**
    $z_i = x_j^{\mathbf{T}} w_i$;
    obtain $\hat{p}_i = \hat{p}_i(z_i)$ from h_i;
  end for
  return $f(x) = -\frac{1}{k} \sum_{i=1}^{k} log\left(\hat{p}_i(z_i)\right)$;

not stand true in practice (Pevný (2016)). However, LODA can still deliver very good results, which may happen for similar reasons as those in naive Bayes classifiers theoretically studied by (Zhang (2004)), which give conditions that cancel out the effects of conditional dependencies. A secondary drawback of this method is that it works only with numerical values as IF and autoencoder. The LODA algorithm is explained in great detail, complemented by the mathematical background in the original paper (Pevný (2016))

Chapter 3

Measuring the anomalies

The goal of this tool, as aforementioned, is to ease the process of looking for outliers for data scientist, thus ideally the user should input the dataset and receive an anomaly score of each tuple which later on can be aggregated to an overall score for the dataset. Then, data scientists can determine whether the dataset would be appropriate for the intended use omitting countless hours spent to wrangle the dataset. In this context, the first step for the user is to provide a valid dataset to the algorithm as well as some parameters which will be discussed in detail later. The next step that occurs is the anomaly detection in each attribute individually and/or in pairs and/or triplets of attributes based on the selected algorithm by the user. Finally, the anomalies are gathered on each tuple and an anomaly score is calculated for each tuple based on the method selected by the user.

3.1 Quantifying the anomaly row-wise

The four aforementioned methods are used to identify the anomalies in the dataset, yet identifying anomalous cells in a dataset is only the first step. Now, the issue of ranking the tuples in anomaly needs to be addressed. For the ranking of the tuples, there are two alternatives, count of occurrence and pagerank. Those two alternatives are different ways to rank the tuples in terms of their anomaly, though this is inadequate as no overall score for the dataset is derived. To attain this, the tuples with highest scores are labelled as outliers and the overall score is the ratio of outlying tuples over the total number of tuples in the dataset.

3.1.1 Count of occurrence

For this method, I go through all the anomalous values iteratively in single, pairs and triplets of anomalies and select a subset where this anomalous value occur in the same attribute as in the initial anomaly. Then, in this subset all the anomalies where the anomalous value occur are counted and assigned as a temporary score for all the tuples of the subset. By the time, that all the anomalous values are checked each tuple has a score based on the number of anomalous values in it as well as the number of anomalies this anomalous value takes part it. According to this method, pairs of attributes anomalies are heavier weighted than single

Algorithm 5 Count anomalies in row-wise ranking

```
for anomalous value do
    select subset with the value in A
    for row in subset do
        if subset[A] = value then
            line score += 1
        end if
    end for
end for
```

Algorithm 6 Page rank algorithm.

Input: nodes $\{n_i\}_1^n$, edges e,timesteps t
Output: Probability of random walker stepping on each node at timestep t

```
initialize Scores
π^(0) ← (1/n, 1/n, ..., 1/n);
R ← β · P + (1 − β) · v · e^T;
while i < t or reaches convergence do
    π^(i+1) ← R · π^(i);
end while
return π^(i)
```

anomalies and triplets of attributes than pairs of attributes. This can be explained by the fact that this iterative method goes through all the anomalous values of a pair, meaning that some anomalies are counted twice, once for each value occurring in the pair of attributes. For an easier interpretation, an extreme example can be pictured of two tuples of 4 attributes in total where one has 4 single outliers and the other one 4 outliers where in all of them one of the attributes takes part it. Both have four outliers in total, where the first one has extreme values in terms of the corresponding range of values whereas in the second tuple the values are not extremely high or low, yet they deviate from the formed pattern of the corresponding combination of attributes. In this occasion the second tuple will achieve a higher anomaly score due to one attribute occurring in all the anomalies, thus all the attributes will be counted twice.

One could argue that this statistical measure is too naïve and, therefore, may not yield good results. A more sophisticated alternative to avoid this statistical metric is the Pagerank algorithm. Before delving deeper into it though, it should be mentioned that is more computationally intensive as a network must be built.

3.1.2 PageRank Algorithm

For this method of ranking, initially a graph needs to be created. The nodes of the graph is the unique anomalous values of each attribute and the edges connect the anomalous value that occur in the same tuple.

The PageRank algorithm consists of two fundamental entities, initial distribution and transition matrix, and using just these entities it assigns a score to each node, hereto anomalous value, indicating how likely it is to occur. This algorithm can be though as a random walker who starts from a random node and in each timestep walks to a random yet connected to the

current node. The higher the score of a node the more likely it is for the walker to move to this node. The initial distribution $\pi^{(t)}$ is a vector of n values, where n is the number of nodes and t is the timestep, which are all equal to 1/n, indicating that in the beginning all the anomalies are equally probable for the walker to step on so that bias is avoided. The transition matrix (P) is a square and positive matrix, $n \times n$, where every a_{ij} in P is given by the relation 3.1.

$$a_{ij} = \sum_{j} \frac{a_{ij}}{d_j}, \tag{3.1}$$

where d_j is the outdegree of the node. In other words, this means that each node share it's score equally to all the nodes that is linked with. In the next timestep $\pi^{(t+1)}$ can be calculated by the formula: $\pi^{(t+1)} = P \cdot \pi^{(t)}$. The $\pi^{(t)}$ shows the probability of the random walker to be on each node. From this formula we can see that $\pi^{(t)}$ is an eigenvector of the matrix P as $\pi^{(t)} = P \cdot \pi^{(t)}$ and the eigenvalue of it is 1. Generally, it would be mandatory to compute all the eigenvectors to find the one describing the graph correctly which would make the process even more computationally intense. However, according to the Frobenius-Perron theorem (reference) there is a unique positive eigenvector for such matrices and in this case is the vector π must be positive (all the consisting elements are probabilities) thus it is the dominant vector with eigenvalue of 1. The algorithm is terminated when it converges to a value according to the power method algorithm (reference). However, there are two issues with convergence in graphs, the first one is the case that the walker reaches a point where can move to one node and back, which is also known as the spider trap problem and the second one is reaching a node which has no out links, which is known as a dead end. To solve these two problems, teleportation is introduced to algorithm which is nothing more than an extra term to alternate slightly the transition matrix in a way that is each timestep there is a slight probability of teleporting the walker to a completely random node. The notion of teleportation can be mathematically expressed as in 3.2

$$\begin{aligned} R &= \beta \cdot P + (1-\beta) \cdot v \cdot e^T \\ v &= (1,1,1,...,1)^T \\ e^T &= \left(\frac{1}{n}, \frac{1}{n}, \frac{1}{n}, ..., \frac{1}{n}\right) \end{aligned} \tag{3.2}$$

, where β is the probability of a normal step for the walker and $1-\beta$ is the probability of teleporting to a random node, v is a vector of ones and e a vector of 1/n, both with length of n.

The product of this algorithm is a score for each node of the graph indicating how likely is for the walker to step on it as explained in algorithm 6 with the help of equation 3.1 and 3.2, which means how anomalous each value is as values which connects to more anomalous values are more anomalous themselves. Then, the anomaly score of each tuple is the sum of the PageRank score of the anomalous value that are contained in the tuple.

As mentioned above, the PageRank algorithm yields goods results for directed graphs, but there are a some instances of it being used in undirected graphs in literature (Abbassi and Mirrokni (2007)), (Andersen et al. (2006)), (Iván and Grolmusz (2011)), (Perra and Fortunato (2008)) and according to (Grolmusz (2015)) it can be used in undirected graphs and the PageRank score is proportional to the degree distribution of the graph.

3.1.3 Ngrams ranking of anomaly

Both of the aforementioned methods require a specific attribute to be anomalous in order to return a proper result. However, this is not the case in the Ngrams, where an outlier is not the cell of an attribute but a part of a string type of data, which is problematic if one wants to utilize those methods, thus a different approach should be used to come up with a score of anomaly for each row. The method followed in this paper is to assign a score to each row based on then number of labelled anomalous n-grams. In detail, the score of each row is the sum of scores for single, pairs and triplets of attributes, which are derived by simply counting the number of labelled anomalous n-grams in the corresponding row. The labelled anomalous n-grams are those which occur fewer times than a percentage of the average (which is set by the user, default is 0.5). Finally, a threshold is set to return the percentage of the most anomalous rows based on the anomaly score of each row, e.g a threshold of 0.2 would return the 20% of highest anomaly score rows.

3.2 Quantifying the anomaly attribute-wise

3.2.1 Quantifying with AE, IF, LODA

Another way to derive an overall score for the dataset is to quantify the anomaly per attribute and then aggregate the scores of the attributes to calculate the overall score.

Initially, the anomalies in each row are identified for the whole dataset based on one of the available algorithms. Afterwards, a score is assigned to each attribute based on the number of times an anomaly occurs containing a value of this attribute. At this point, each attribute has a score assigned to it, though this number is not indicative of anomaly of the attribute in comparison to the rest of the dataset. Therefore, normalization is needed in the scores so they are comparable. The normalization factor is the number of the anomalies an attribute would potentially participate in if every value of it was an outlier, which is given by the formula 3.3.

$$nfactor = n \cdot (1 + (m-1) + (m-1) \cdot (m-2)) = n \cdot (1 + (m-1)^2), \tag{3.3}$$

where n is the number of rows and m is the number of attributes of the dataset. The first factor is about the attribute participating in a single outlier, the second one is about participating in an anomalous pair and the third one about a triplet.

Since all the attributes are assigned with a normalized score, hence they are comparable, the overall score of the dataframe can be attained by aggregating the attribute scores, e.g. by calculating the mean of the scores. Even though, different functions can be used to aggregate the scores such as median of the scores, hereto the mean of them is primarily used.

3.2.2 Quantifying with Ngrams

Similarly to row-wise quantifying of anomaly a different approach is used in Ngrams algorithm due to the different nature of outlier detection. Initially, a score is assigned to each

attribute which is derived by the number of labelled outlying Ngrams that occur in a value of the corresponding attribute. However, this score is not comparable between different attributes as the number of maximum potential anomalous Ngrams in an attribute is strongly correlated with the length of the string type data. Thus, each attribute score is devided by the potential maximum number of anomalous ngrams in it. This upper limit is calculated with formula 3.5, which is derived by summing 3.4 .

$$\begin{aligned} nfactor_{single} &= \sum_{i}^{\#rows} (length(value) - n + 1)_i, \\ nfactor_{pair} &= \sum_{i}^{\#rows} (length(value_1 + value_2) - n + 1)_i, \\ nfactor_{triplet} &= \sum_{i}^{\#rows} (length(value_1 + value_2 + value_3) - n + 1)_i, \end{aligned} \tag{3.4}$$

where n is the length of the Ngram and $length(value_i)$ is the number of characters in the cell of an attribute.

$$\begin{aligned} nfactor &= nfactor_{single} + nfactor_{pair} + nfactor_{triplet} = \\ &= \sum_{i}^{\#rows} (length(value) - n + 1)_i + \sum_{i}^{\#rows} (length(value_1 + value_2) - n + 1)_i + \\ &\quad + \sum_{i}^{\#rows} (length(value_1 + value_2 + value_3) - n + 1)_i. \end{aligned} \tag{3.5}$$

At this point, each attribute has a normalized anomaly score assigned to itself, so anomaly in attributes now can be compared and an overall score can be calculated for a dataset by aggregating the attribute scores by any aggregating function, though in this paper the mean is used primarily.

At this point, the tool has in its arsenal 4 different approaches to detect the anomalous points and 3 ways to rank the attributes or tuples of the dataset based on the anomaly score and can return an overall anomaly score for the dataset. However, the performance of some of the detection methods strongly depends on the inputted hyper parameters while initializing the method, thus fine tuning of those parameters is needed in different situations so the overall performance of the tool is optimized.

3.3 Process of the tool

As it has already been mentioned, this is the first tool that gives the opportunity of having four different approaches while combining it with three methods to rank the anomaly in the dataset and return a single score. To ease the process of familiarizing with the tool, in this section all the option are explained.

In figure 3.1, there is the over view of tool. Initially, the user needs to input the dataset

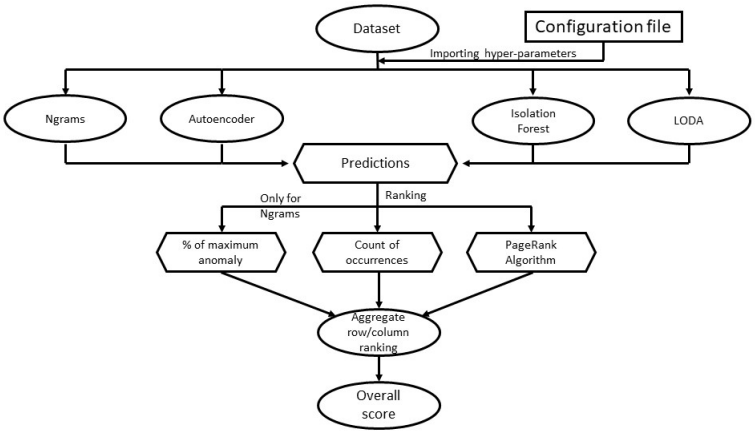

Figure. 3.1: Overview of the outlier detection tool.

and based on the size of the dataset the corresponding hyper-parameters are set based on the configuration file. Consequently, the user needs to choose one of the available detection methods (Ngrams, Autoencoder, Isolation Forest, LODA) and determine the training split for the methods that need training or the size of the Ngrams and the threshold for outliers in the Ngrams. This threshold is the percentage of deviance from the mean frequency of all the ngrams up to which is considered acceptable. Afterwards, ranking of the anomalies needs to be done, which can be achieved in two ways, row-wise and attribute-wise. At this point, one more threshold is needed, which indicates the elements that will be considered as outliers depending on the ranking score.

In row-wise ranking, the PageRank algorithm can be utilized or a simple count of the anomalies each anomalous value occurs in and based on the anomalies each tuples contains (summed anomalies each anomalous value participates in the corresponding row), they are ranked in terms of anomaly. The summary of the simple count of the anomalies in each tuple can be read in the form of pseudo code in algorithm 5. The Ngrams method follows a different approach of ranking as it is based on counting the number of anomalous Ngrams that occur in each row. In attribute-wise ranking for all the methods except from the Ngrams, the number a value of each attribute participates in an outlier is counted for each attribute separately and these scores are normalized by a factor as described in formula 3.3. For the Ngrams method, the score of each attribute is measured as the ratio of anomalous Ngrams over the potential maximum Ngrams. Finally, the overall score for the row-wise and attribute-wise is the percentage of anomalous rows over all the rows and the aggregated attribute scores, which can be aggregated by the mean or the median.

Chapter 4

Implementation and Installation

4.1 API for the created library

This tool can be used directly from the python terminal. Although, this can be useful on its own, it would be also practical if it could be included in any script for individual use, hence a library has been created to enable users to use it more conveniently. This library can be imported by using `import outlier_detection_library` as `od` and contains to classes, the Detector and the Ranker.

Detector objects just need one parameter to be determined for them to be initialized, which is the `method` parameter indicating which detection method will be followed to determine which data points are outliers and its default value is LODA as it is a fast and accurate method. Detector objects give the option to the user to import data from a local file using the path of it with the `get_data` method, which is an improved of the `read_csv` and `read_excel` offered by pandas as it tackles some of the limitations of the aforementioned methods such as the reverse slash (Windows users) and also can read both .csv and .xlsx files. The second method of these objects is the `detect` method which needs a few parameters to be set such as the dataset of interest, the `train_split` parameter (ratio of dataset used for training), `con` parameter (ratio of outliers in the dataset) which can take values between 0 and 0.5, `n` parameter which determines the length of the grams in case Ngrams method is used for detection and, finally, the `th` parameter which set the limit of acceptable deviance from the mean frequency in Ngrams for a point to not be labelled as an outlier. The default values for these parameters are 0.5, 0.1, 3, 0.3 respectively.

The Ranker objects need more parameters to be initialized, which are `method`, `att_ranking`, `PageRank` and the `threshold` parameter. The `method` parameter is the same as the one in the Detector objects, the `att_ranking` parameter determines whether the ranking of the dataset will be executed in the rows or in the columns of the dataset and it is a binary parameter, where 0 means that the ranking occurs in the rows whereas 1 means that the ranking occurs in the columns. The `PageRank` parameter is binary as well, where 0 means the the count method which is explained in 3.1.1 and 1 that the PageRank algorithm is initialized. The final parameter is the `threshold` parameter which sets the lower limit of the interval of anomaly scores which will be labelled as outliers. For example, let the `threshold` parameter be 0.1, and the range of the

anomaly scores in a conjectural dataset be $[100 - 1000]$, then data points with score greater than $1000 - 0.1 \cdot 900$ will be labelled as outliers. The default values for these parameters are `loda`, 0, 0, 0.1 respectively. The Ranker objects offer the `rank` method which needs three parameters to function properly, which are a dataframe with the predicted outliers in the same format as the one provided by `detect` method of Detector objects, the dataset of interest and the `function` parameter which indicates the aggregating function to be used to get the overall score of the dataset based on the scores of either rows or columns of the dataset. The available values for this parameter is `mean` and `median` and the default one is `mean`.

4.2 Overview of the tool

This tool is composed of three components. The most crucial component of the tool is the outlier detection tool, which based on the given dataset as well as the instructions by the user can make predictions about which values of the dataset deviate significantly enough to be considered outliers. Consequently, the ranking component based on the outliers of each row ranks the rows from the most to the least anomalous and returns to the user those with the highest anomaly score. Finally, the last component of the tool is the parser, which essentially the user who imports the path of the dataset. At this point, it should be mentioned that a configuration file is used before the detection component is initialized to import the hyper parameters needed for it to work.

4.3 Installation

4.3.1 Hyper-parameter tuning

The two algorithms that need to be tuned are the Autoencoder and the Isolation Forest. The hyper-parameters for the former algorithm that are tuned are the number of epochs that that the model is trained, the activation function that is used in the hidden layers, the number of the hidden layers, the number of neurons per hidden layer and the batch size. The hyper-parameter tuning is done with grid search cross validation, which means that a 4 dimensional grid (number of hidden layers and neurons in them are compressed into one dimension) is created with the values of interest for each dimension and then for each combination of parameters cross-validation occurs. For both datasets and all the settings, a 90% split is used for training and the rest 10% is used for validation. Cross-validation is repeated 3 times so the tuning is statistically more robust. Last but not least, the metric for calculating the score of each set of parameters is the accuracy, which is defined as the $acc = (TP + TN)/(TP + FP + TN + FN)$, where TP, TN is true positive, true negative respectively.

The search grid of the autoencoder method consists of 4 dimensions. The first one is the batch size which can be 32, 64 or 128, the second one is the number of epochs which can be 20, 50 or 100, the third one is the number of hidden layers and the number of neurons in each of those layers. This can be attained by importing a list of n elements, where n is the number of hidden layers and each element is an integer which represents the number of neurons in the layer and

the available option on the grid are [64,32,16,1,16,32,64],[32,16,1,16,32] and [4,1,4]. The choice of the number of layers and neurons is the one that determines whether the model will over- or underfit in the training data, thus the three given option having gradually decreasing complexity both in number of layers and neurons per layer. The fourth dimension is the activation function of the hidden layers which can be either 'relu' or 'tanh'. In short, the 'relu' function returns 0 if the input is lower than a specified threshold (usually 0) and linear with gradient of 1 for inputs greater that the threshold. The 'tanh' function returns the value of the hyperbolic tangent of the input. Both of them are used to ensure that the the output of a hidden layer will not be linear, thus more complex patterns can be detected. The search grid of the Isolation Forest method consists of just 1 dimension which is the number of the binary trees and the values of it for the fine-tuning is 50,100,200 and 400.

Before diving into the fine-tuning of those two methods, it should be mentioned that both methods depend on a hyper parameter called contamination which affect significantly the results of these algorithms as it determines the threshold, which if exceeded a point is labelled as an outlier. This parameter should be set based on the percentage of outliers in the dataset, which requires the user to know it in advance, which in most cases is a fault assumption. Thus, in the hyper parameter tuning, even though the contamination percentage of both datasets is known, this parameter will remain is the default values which is 0.1 as in real-world situation even if the user can change it, it would be a rough estimation instead of the actual value.

Autoencoder fine-tuning

For the Vertebral dataset, which is a small dataset containing real-world data the optimal hyper-parameters are 20 epochs of training, batch size of 128 inputs per batch, 'tanh' as the activation function in hidden layers and a network which consists of 7 layers with 64 neurons in the first hidden layer which are step-wise halved and a bottleneck of 1 neuron (bottleneck is explained later one).

For mulcross dataset, which is a vast dataset containing synthetic data the optimal hyper-parameters are 100 epochs of training, batch size of 32 inputs per batch, 'relu' as the activation function in hidden layers and a network which consists of 5 hidden layers with 32 neurons in the first hidden layer which are step-wise halved and a bottleneck of 1 neuron as it can be seen in table 4.1.

A second fine-tuning has been done and the results are slightly different for the autoencoder as for the mulcross dataset the batch size is reduced to 32, but more interestingly the complexity of the model increased as the hidden layers are 7 with the first one containing 64 neurons while the activation function in the hidden layers remained the same. In the Vertebral dataset even though there were changes in the parameters such as reduced hidden layers, only 5 with the first one containing 32 neurons, and different activation function, 'relu' is picked this time, the complexity of the model is not significantly altered as a more complex activation function is chosen, yet the reduced hidden layers compensate for this increase in complexity. The batch size for the latter dataset remained steady and the number of epochs for both dataset, which indicates that the slight deviation between the two runs may be due to the inherent randomness

a/a	Autoencoder				
	Batch size	Epochs	Hidden function	Neurons/ Layers	Avg time
Mulcross	32	100	relu	[32, 16, 1, 16, 32]	46028s
Vertebral	64	20	tanh	[64, 32, 16, 1, 16, 32, 64]	265s

Table 4.1: Fine-tuning for Autoencoder algorithm.

a/a	Isolation Forest	
	Estimators	Average time
Mulcross	50	250s
Vertebral	400	7s

Table 4.2: Fine tuning for Isolation Forest algorithm.

of the autoencoder.

Isolation Forest fine-tuning

For the Isolation Forest the hyper parameter tuning is remarkably simpler. There is only one parameter that needs to be fine tuned, which is the number of the estimators which are used to determine the number of the binary trees which will be trained in the ensemble.

The results of the fine-tuning of the Isolation Forest model are the following: for the extensive mulcross synthetic dataset only 50 binary trees are enough to yield the best possible results out of all the available options. On the other hand, the small dataset with real-world data needs the maximum of the available numbers of estimators which is 400 binary trees as it can be seen in table 4.2.

Chapter 5

Experimental evaluation

The proposed algorithm is tested on two different datasets. There are two reasons for this testing, the first one is that, obviously, performance tests are needed to evaluate the algorithm and the second one is that they are some hyper-parameters in some algorithms that are initialized which need to be tuned.

5.1 Data

Those two datasets are the Vertebral dataset and the Mulcross dataset. The former dataset is found on the machine learning repository (https://archive-beta.ics.uci.edu/ml/datasets/vertebral+column) Barreto (2011) and it consists of 240 instances and 6 attributes. Each instance refers to a different patient and the patient is represented by six biomechanical attributes which are pelvic incidence, pelvic tilt, lumbar lordosis angle, sacral slope, pelvic radius and grade of spondylolisthesis, though the units of these measurements are not given by the creators. The latter is found OpenML https://www.openml.org/search?type=data&sort=runs&id=40897&status=active. The Mulcross dataset consists of 262144 rows and 5 columns, one of which is the label of outlier or not. What is interesting about this dataset is that it is generated from a synthetic data generator Mulcross, which basically generates a multi-variate normal distribution with a user specified number of anomaly clusters. The details of this specific dataset can be found in Liu et al. (2012), but in short the ratio of anomalies to total number of points is 10%, the dimensions are set to 4 and the distance factor, indicates the distance between the center of normal cluster and the centers of anomaly clusters, is set to 2. Since, this dataset consists of synthetic data, no information is given regarding the attributes apart from the fact that all of them have numeric values.

The reasons behind the selection of these datasets may be quite obvious, yet they need to be mentioned. Initially, having a dataset with real data and one with synthetic data gives the opportunity to make a comparison of performance of the algorithms between real-world and synthetic data. Furthermore, both of the datasets have single digit number of attributes which is helpful as limit computational power is available. Even a negligible increase in the number of attributes, will exponentially increase the complexity of the problem as a model needs to be

Table 5.1: Accuracy, precision and recall of all methods on Vertebral dataset with the threshold set to 0.3.

Detecting method	Ngrams	AE	AE	IF	IF	LODA	LODA
Ranking method	-	Count	Pagerank	Count	Pagerank	Count	Pagerank
Accuracy	0.85	0.87	0.70	0.825	0.57	0.85	0.79
Precision	0.125	0	0.004	0	0.06	0	0
Recall	0.003	0	0.007	0	0.16	0	0

trained for all the possible combinations of attributes separately and then make predictions. For example, an increase from 6 to 7 attributes would increase the possible combinations of single attribute, pairs of attributes and triplets of attributes by 22, which means that 22 more models need to be trained, which translates to an immense increase in computational load.

A similarity that both of these datasets share, is that each of their tuple comes labelled as an inlier or outlier in the attribute 'label' and 'class', respectively. This attribute is excluded in the process of outlier detection and retrieving the anomaly score of the dataset, though it is used later to evaluate the performance of different algorithms. Also, the same datasets can be used to tune the hyper-parameters of some of the algorithms and conclude which hyper-parameters work best for real-world and synthetic data.

5.2 Quality of the results

In this work, the evaluation of the results of the tool will be done on the Vertebral dataset only, for reasons explained later on in detail. All the points of the dataset will be labelled either as inliers or outliers and the metrics used for evaluation are accuracy, precision and recall, which are given by $pr = TP/(TP+FP)$ and $Re = TP/(TP+FN)$ respectively for precision and recall, where TP is true positive and TN is true negative. Also, the threshold is set for the test at 0.3. The accuracy formula is given in 4.3.1 section. Precision indicates the percentage of accurately labelling a data point as an outlier out of all the actual outliers, whereas recall indicates the percentage of truly outlying points detected by a method as outliers. According to table 5.1, the accuracy of all the detecting methods is quite high, whereas the precision and recall metrics are almost always below 10%. The high accuracy of all the algorithms can explained by the fact that inliers, which is the majority of the data points, are mostly correctly labelled though the detection methods face a difficulty with identifying points that deviate from the pattern, hence the low precision which peaks at 0.125 using the Ngram method. Additionally, even for the points labbeled as outliers the percentage of them actually being outliers is quite low with the IF in combination with the Pagerank algorithm to rank the rows of the dataset reaching a peak of 0.16. Another important point is that both ranking methods in combination with LODA have 0 precision and recall. These mediocre results can be explained by the low threshold that has been set. Therefore, an analysis about the threshold value should be done to determine the results each method yield for this specific dataset.

Through the threshold value analysis, some conclusions can be drawn regarding the two different ranking methods. For the figures 5.1, 5.2, 5.3 only the metrics for the AE method

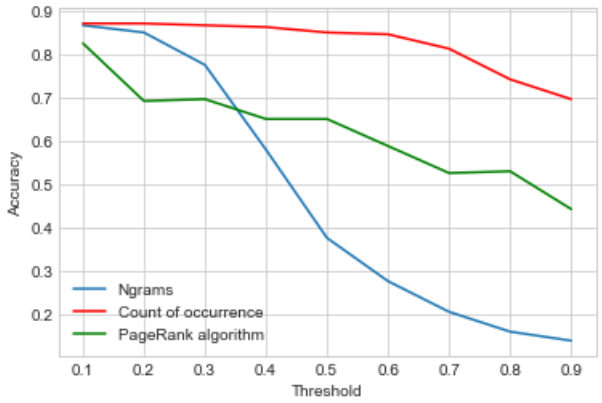

Figure. 5.1: Accuracy of the Autoencoder method and the Ngrams method against the threshold value.

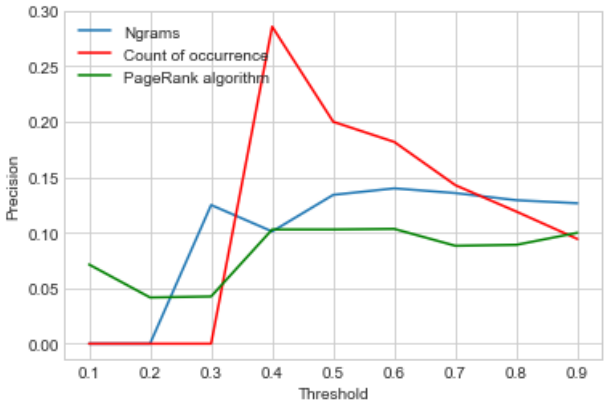

Figure. 5.2: Precision of the Autoencoder method and the Ngrams method against the threshold value.

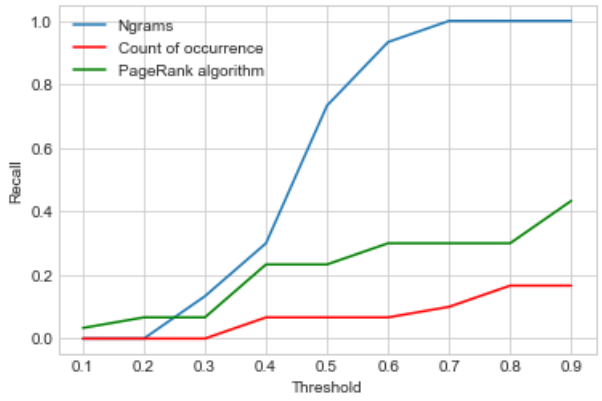

Figure. 5.3: Recall of the Autoencoder method and the Ngrams method against the threshold value.

are shown since the metrics for all the methods demonstrated similar behaviour in the same range of values and it would just make the graph more challenging to interpret. Initially, for all the ranking methods the accuracy seems to reduce as the threshold value increases, and for the Ngrams method it seems that for threshold values greater than 0.7 it plummeted below 0.2. Also, regarding the accuracy the count of occurrence method yields approximately 20% better results that the Pagerank algorithm. Similar behaviour can be seen in the precision, where the count of occurrence yields up to 3 times better precision than Pagerank algorithm when the threshold value is set to 0.4. However, in figure 5.3 can be seen that the Pagerank algorithm yields consistently better results, which can be up to twice as good as the count of occurrence method for high threshold values such as 0.9. In the same figure, it is outstanding that the Ngrams method can reach recall of 1 and it even converges to that value for values of threshold greater than 0.7.

5.3 Limitations

Even tough this tool yields satisfactory results, it still suffers from a few limitations. Initially, LODA is the fastest and scales more efficiently in big data than the four methods and yields as good if not better results as the rest of them, though it should not be forgotten that it functions under the assumption that the attributes are independent which in most cases does not stand true. Thus, a test for correlation between the attributes should be run before using this method. Another limitation of the tool is that ranking by the PageRank algorithm is a slow process. Specifically, when it was used with the AE on a system with an AMD Ryzen 3900X and 16GB of RAM running Windows for the detection and evaluation of the mulcross dataset (approx. 250000 instances), the detection of the outliers with AE, which is a relatively slow method as a deep network needs to be trained, took marginally longer than an hour and the ranking with the PageRank algorithm after 17 hours of running was not complete yet.

5.4 Performance

The performance of the tool is discussed in the following part, in terms of time needed to execute the whole process as well as scalability. The aforementioned system setup is used for the evaluation of the performance. All algorithm were used on the Vertebral dataset separately and, then, all the ranking methods were tested as well. In table 5.2 the run time of each method for detecting and ranking can be seen. The ranking process for the Ngrams method is almost instantaneous as it took only 0.001 seconds but is not stated table 5.2 as it does not follow any of these two tanking methods. According to this table, the method utilizing Ngrams is significantly the slowest one, and is followed by the AE which is approximately 4 times faster. The remaining two methods, are quite faster than AE which IF being done after 17.07 seconds and LODA after just 1.1 second. These results, are expected as IF and LODA, do not need a network to be build hence the reduced run time. Also, LODA was expected to be the fastest one of them all as it is used for streaming high volume weather data as aforementioned. On the other hand, what is remarkable is the run time of Ngrams as no training is needed and the process of

Table 5.2: Run time of different detecting and ranking methods measured in seconds.

Detecting/Ranking	Ngrams	Autoencoder	Isolation Forest	LODA
Detection	206.37	51.01	17.07	1.1
Count of occurences	-	1.25	2.46	1.25
PageRank algorithm	-	9.18	26.51	7.95

tokenizing and calculating some statistical metrics is fairly fast. Furthermore, according to the table 5.2 PageRank algorithm, as expected, is more computationally intensive and according to this performance test is on average 8.13 times slower than the count of occurrences.

As aforementioned, the PageRank algorithm was not completed after 17 hours of running. This can be explained by the fact that the dataset consists of approximately a million values, thus the number of single as well as pairs and triplets of outliers are numerous which leads to an extensive networks as each outliers is represented as a vertex in the network. As a result, this network maybe is not even possible to be created due to shortage of RAM memory and the process was stuck at a certain point. Therefore, the results and quality of results of the Mulcross dataset have not been mentioned in this work.

If one takes into consideration the time needed to run a single AE network looking at table 4.1and the volume of data, which is approximately 903 times larger in the Mulcross dataset than in the Vertebral dataset, it can be deduced that the AE scales sublinearly with the volume of data as it needed only 173.7 more time to complete a single network in the Mulcross than in the Vertebral datset. Again, the sublinear scaling of the algorithm holds true for these two datasets with the specific underlying complexity of the data. Also, it should be taken into consideration that more neurons and layers were initialized in the less extended dataset as well as fewer epochs were needed in the training process, which are two factors that counter act each other in the time complexity of the model, so minor deviation due to those two factors is expected.

Moreover, IF is outstandingly faster than AE in both datasets, approximately 200 times in the Mulcross and 40 times in the Vertebral dataset faster than AE and it even scales better than AE as it needs only 35 more time for an increase in volume of data of 903. The difference in speed of these two algorithms can be explained by the different approach they follow to detect outliers as selecting random value and splitting a dimensions is simpler and faster for the cpu than solving differential equations which is needed for the backpropagation of error in training process of deep networks.

5.5 Discussion

Due to the computational load restriction conclusion can all be drawn from the hyper-parameter tuning of the algorithms. Since the fine-tuning give quite different optimal parameters for the two datasets, it can be deduced that the models need to be tuned based on the different nature of the datasets. In detail, small datasets such as the Vertebral dataset with 6 attributes only and 240 instances are benefited more by larger batch size as the highest available number of batch size was chosen and more complex deep networks as the best performing one was the one

with 7 layers where the most complex layer had 64 neurons. It counter intuitive that, although, high complexity in the structure of the network is chosen, the number of epochs is the lowest available which indicates that overfitting would occur for a higher number of epochs. Finally, the activation function that is used is the not so advanced 'tanh' function and, apparently, the 'vanishing gradient problem', as proposed by Hochreiter (1998), does not appear in so small datasets. It should be mentioned that even though fine-tuning is properly done, the number of neurons and layers of the networks is strongly correlated with the underlying complexity of the data, meaning that number of neurons and layers may variate according to the complexity of the data. On the other hand, in the artificial mulcross dataset, which is strikingly more extended, the lowest available batch size is selected and a simpler structure of network with only 5 layers with the first one having 32 neurons is chosen. However, 100 epochs are needed for the loss (error) to converge. The activation function is the 'relu' function. Another point that should be mentioned is that the set of 3 hidden layers with only 4 neurons and the bottleneck of 1 neuron is too simple to detect any pattern in the data, even for the synthetic dataset which was build based on a multi-variate normal distribution generator. Those results can be seen in table 4.1 as well as the average time of running a model in the hyper parameter tuning.

For the IF algorithm the difference in the number of estimators for the two datasets may be explained by the fact that due to the shortage in training data in the Vertebral dataset more estimators are needed to reduce bias in the predictions, whereas for the artificial dataset where plenty of data is available for the training process a simpler model with fewer estimators is the optimal option for detecting outliers.

Chapter 6

Conclusion

6.1 Key Points

To sum it up, this outlier detection tool is quite useful in the hands of data scientists as it can save a lot of valuable time by yielding good results regarding the number of outliers in a given dataset as the methods of detection initialized in it have been proved to successfully detect anomalies. Furthermore, the anomaly score can be calculated either by rows or by attributes giving the option to the user to determine whether some data points are anomalous to be taken into consideration in the analysis or a whole attribute could may cause more harm than good in the analysis. To accomplish that, three different approaches are offered to rank the tuples/attributes of the dataset depending on the preference of user as well as the chosen method of detection. To be even more efficient, the output of this tool is a single overall score of the dataset based on the parameters that were selected, yet the most anomalous data points are outputted as well in case the user wants to further examine those outliers. Finally, this tool can be executed through the python terminal so one can have an expeditious result and an independent library has been created to ensure that anyone can import and further process the output of it. However, it should not be forgotten that this tool expects a dataset containing no missing values or error values (e.g. -100K for temperature in case the sensor fails) as it would either return an error or try to learn a pattern based on the error values. Therefore, appropriate pre-process of the dataset is needed prior to the usage of the tool.

In this work there is no evaluation of the outlier detection tool on real-world data and comparison between the result produced by all the algorithms in terms of accuracy, precision and recall. The primary reason behind this deficiency of this work is the limitation of the computational power as datasets containing real world data usually have hundreds of thousands of instances with numerous attributes, usually more than ten, which make it infeasible for a typical desktop computer to run so complex algorithms such as AE or PageRank algorithm on so many instances. Furthermore, the ideal comparison would be to run the Ngrams method with one of the remaining methods each time and measure the average accuracy, precision and recall in each run. This should be done because usually datasets contain both string type and float or integer type of data. After the three runs the accuracy, precision and recall as well as the time of each run can be compared to determine which combination of methods can yield

the best results for the chosen dataset. Later on, the same process could be executed but for datasets with different properties, different volume of data or synthetic data so conclusions could be drawn for which algorithms are more efficient for each type of dataset.

6.2 Future work

As mentioned previously, the results fine-tuning of the algorithms seem to deviate slightly from run to run, which couldn't be further examined due to limited computational power. Hence, it would be interesting to see the results of fine-tuning with more available options for all the parameters as well as more datasets to make the comparison of their characteristics even more unambiguous. For instance, two datasets can be used with same type of data (synthetic/ real-world) and different size, and similarly with just different type of data as well as different size in the same type of data to examine the scalability of the tool. Furthermore, as explained in Chapter 3.1.3 the Isolation Forest algorithm has some inherent bias, which may be addressed by manually increasing the anomaly score in the rectangles surrounding the anomalous area.

Bibliography

Zeinab Abbassi and Vahab S. Mirrokni. A recommender system based on local random walks and spectral methods. In Haizheng Zhang, Myra Spiliopoulou, Bamshad Mobasher, C. Lee Giles, Andrew McCallum, Olfa Nasraoui, Jaideep Srivastava, and John Yen, editors, *Advances in Web Mining and Web Usage Analysis, 9th International Workshop on Knowledge Discovery on the Web, WebKDD 2007, and 1st International Workshop on Social Networks Analysis, SNA-KDD 2007, San Jose, CA, USA, August 12-15, 2007. Revised Papers*, volume 5439 of *Lecture Notes in Computer Science*, pages 139–153. Springer, 2007. doi: 10.1007/978-3-642-00528-2_8. URL https://doi.org/10.1007/978-3-642-00528-2_8.

Charu C. Aggarwal. *Outlier Analysis*. Springer Publishing Company, Incorporated, 2nd edition, 2016. ISBN 3319475770.

Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Local graph partitioning using pagerank vectors. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 475–486. IEEE Computer Society, 2006. doi: 10.1109/FOCS.2006.44. URL https://doi.org/10.1109/FOCS.2006.44.

Ajalmar Barreto, Guilherme Neto. Vertebral Column. UCI Machine Learning Repository, 2011.

Divya D. and Dr Sasidhar Babu. Methods to detect different types of outliers. pages 23–28, 03 2016. doi: 10.1109/SAPIENCE.2016.7684114.

Ralph Foorthuis. On the nature and types of anomalies: a review of deviations in data. *Int. J. Data Sci. Anal.*, 12(4):297–331, 2021. doi: 10.1007/s41060-021-00265-1. URL https://doi.org/10.1007/s41060-021-00265-1.

Vince Grolmusz. A note on the pagerank of undirected graphs. *Inf. Process. Lett.*, 115(6-8):633–634, 2015. doi: 10.1016/j.ipl.2015.02.015. URL https://doi.org/10.1016/j.ipl.2015.02.015.

Sahand Hariri, Matias Carrasco Kind, and Robert J. Brunner. Extended isolation forest. *IEEE Trans. Knowl. Data Eng.*, 33(4):1479–1489, 2021. doi: 10.1109/TKDE.2019.2947676. URL https://doi.org/10.1109/TKDE.2019.2947676.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.*, 6(2):107–116, 1998. doi: 10.1142/S0218488598000094. URL https://doi.org/10.1142/S0218488598000094.

Gábor Iván and Vince Grolmusz. When the web meets the cell: using personalized pagerank for analyzing protein interaction networks. *Bioinform.*, 27(3):405–407, 2011. doi: 10.1093/bioinformatics/btq680. URL https://doi.org/10.1093/bioinformatics/btq680.

Ping Li. Very sparse stable random projections for dimension reduction in *l*alpha (0 <alpha<=2) norm. In Pavel Berkhin, Rich Caruana, and Xindong Wu, editors, *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*, pages 440–449. ACM, 2007. doi: 10.1145/1281192.1281241. URL https://doi.org/10.1145/1281192.1281241.

Zhuowei Li, Amitabha Das, and Sukumar Nandi. Utilizing statistical characteristics of n-grams for intrusion detection. In *2nd International Conference on Cyberworlds (CW 2003), 3-5 December 2003, Singapore*, pages 486–493. IEEE Computer Society, 2003. doi: 10.1109/CYBER.2003.1253494. URL https://doi.org/10.1109/CYBER.2003.1253494.

Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 413–422. IEEE Computer Society, 2008. doi: 10.1109/ICDM.2008.17. URL https://doi.org/10.1109/ICDM.2008.17.

Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Trans. Knowl. Discov. Data*, 6(1):3:1–3:39, 2012. doi: 10.1145/2133360.2133363. URL https://doi.org/10.1145/2133360.2133363.

Junaid Maqsood, Iman Eshraghi, and Syed Sarmad Ali. Success or failure identification for github's open source projects. In Yulin Wang, editor, *Proceedings of the 2017 International Conference on Management Engineering, Software Engineering and Service Sciences, ICMSS '17, Wuhan, China, January 14-16, 2017*, pages 145–150. ACM, 2017. doi: 10.1145/3034950.3034957. URL https://doi.org/10.1145/3034950.3034957.

Nicola Perra and Santo Fortunato. Spectral centrality measures in complex networks. *Phys. Rev. E*, 78:036107, Sep 2008. doi: 10.1103/PhysRevE.78.036107. URL https://link.aps.org/doi/10.1103/PhysRevE.78.036107.

Tomás Pevný. Loda: Lightweight on-line detector of anomalies. *Mach. Learn.*, 102 (2):275–304, 2016. doi: 10.1007/s10994-015-5521-0. URL https://doi.org/10.1007/s10994-015-5521-0.

Daphne R. Raban and Avishag Gordon. The evolution of data science and big data research: A bibliometric analysis. *Scientometrics*, 122(3):1563–1581, 2020. doi: 10.1007/s11192-020-03371-2. URL https://doi.org/10.1007/s11192-020-03371-2.

Inka Saarinen. Adaptive real-time anomaly detection for multi-dimensional streaming data. Master's thesis, Aalto University. School of Science, 2017. URL http://urn.fi/URN:NBN:fi:aalto-201704133534.

Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In Ashfaqur Rahman, Jeremiah D. Deng, and Jiuyong Li, editors,

Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis, Gold Coast, Australia, QLD, Australia, December 2, 2014, page 4. ACM, 2014. doi: 10.1145/2689746.2689747. URL https://doi.org/10.1145/2689746.2689747.

Asmaa Sallam, Daren Fadolalkarim, Elisa Bertino, and Qian Xiao. Data and syntax centric anomaly detection for relational databases. *WIREs Data Mining Knowl. Discov.*, 6(6):231–239, 2016. doi: 10.1002/widm.1195. URL https://doi.org/10.1002/widm.1195.

Dong Xu, Yanjun Wang, Yulong Meng, and Ziying Zhang. An improved data anomaly detection method based on isolation forest. In *10th International Symposium on Computational Intelligence and Design, ISCID 2017, Hangzhou, China, December 9-10, 2017 - Volume 2*, pages 287–291. IEEE, 2017. doi: 10.1109/ISCID.2017.202. URL https://doi.org/10.1109/ISCID.2017.202.

Richard Zak, Edward Raff, and Charles Nicholas. What can n-grams learn for malware detection? In *12th International Conference on Malicious and Unwanted Software, MALWARE 2017, Fajardo, PR, USA, October 11-14, 2017*, pages 109–118. IEEE Computer Society, 2017. doi: 10.1109/MALWARE.2017.8323963. URL https://doi.org/10.1109/MALWARE.2017.8323963.

Tadesse Zemicheal and Thomas G. Dietterich. Anomaly detection in the presence of missing values for weather data quality control. In Jay Chen, Jennifer Mankoff, and Carla P. Gomes, editors, *Proceedings of the Conference on Computing & Sustainable Societies, COMPASS 2019, Accra, Ghana, July 3-5, 2019*, pages 65–73. ACM, 2019. doi: 10.1145/3314344.3332490. URL https://doi.org/10.1145/3314344.3332490.

Harry Zhang. The optimality of naive bayes. In Valerie Barr and Zdravko Markov, editors, *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, Miami Beach, Florida, USA*, pages 562–567. AAAI Press, 2004. URL http://www.aaai.org/Library/FLAIRS/2004/flairs04-097.php.

Chong Zhou and Randy C. Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 665–674. ACM, 2017. doi: 10.1145/3097983.3098052. URL https://doi.org/10.1145/3097983.3098052.