Unsupervised Anomaly Detection using K-Means Clustering on a Car Dataset

Kanika Saini
Mechanical and Automation Engineering
Indira Gandhi Delhi Technical University for Women
Kashmere Gate, New Delhi, Delhi 110006
Email: kanika035btmae22@igdtuw.ac.in

Pooja
Mechanical and Automation Engineering
Indira Gandhi Delhi Technical University for Women
Kashmere Gate, New Delhi, Delhi 110006
Email: pooja050btmae22@igdtuw.ac.in

Manisha Sharma
Computer Science and Engineering
Gd Goenka University
Sohna Road, Gurgaon, Haryana
Email: manishacheen@gmail.com

Abstract:

This study explores the application of unsupervised anomaly detection using K-Means clustering on a comprehensive car dataset. Anomalies in the automotive domain can lead to safety hazards and financial losses. Leveraging the K-Means clustering algorithm, we demonstrate its efficacy in identifying subtle deviations from normal behavior within the dataset, thereby aiding in the detection of potential faults or irregularities in car performance. The dataset comprises a diverse range of features, including vehicle specifications, sensor data, and maintenance records. Our results showcase the effectiveness of K-Means in detecting anomalies, revealing its potential for real-world applications in automotive quality control and predictive maintenance. Furthermore, the study discusses the practical implications and challenges associated with implementing unsupervised anomaly detection in the automotive industry, emphasizing its significance in ensuring vehicle safety and reliability

.

1. INTRODUCTION

Anomaly detection also known as outlier detection. It is the process of finding data points within a dataset that differs from the rest. The identification of rare and potentially disruptive events within datasets, is a fundamental task in various domains. Common applications of anomaly detection include fraud detection in financial transactions, fault detection and predictive maintenance. From fraud detection in financial transactions to fault detection in industrial machinery, the ability to uncover hidden anomalies is critical for ensuring system integrity and security.

Anomaly detection can be categorized into supervised and unsupervised realm. Supervised anomaly detection requires labelled dataset that indicates if a record is “normal” or “abnormal”. Unsupervised anomaly detection involves an unlabeled dataset. It assumes that the majority data points in the unlabeled dataset are “normal” and it looks for data points that differs from the “normal” data points.

Supervised Anomaly Detection: This method requires a labeled dataset containing both normal and anomalous samples to construct a predictive model to classify future data points. The most commonly used algorithms for this purpose are supervised Neural Networks, Support Vector Machine learning, K-Nearest Neighbors Classifier, etc.

Unsupervised Anomaly Detection: This method does require any training data and instead assumes two things about the data ie Only a small percentage of data is anomalous and Any anomaly is statistically different from the normal samples. Based on the above assumptions, the data is then clustered using a similarity measure and the data points which are far off from the cluster are considered to be anomalies.

Despite their importance, existing unsupervised anomaly detection methods face challenges in handling high-dimensional and complex data, such as network traffic patterns, sensor readings, and medical records. As a result, there is a growing need for innovative approaches that can adapt to evolving data landscapes.

Anomaly detection in mechanical engineering is a critical application that helps identify unusual patterns or deviations from normal operation in various mechanical systems. It is used for predictive maintenance, quality control, safety monitoring, and improving overall system reliability. There are some key areas where anomaly detection is applied in mechanical engineering: -

1.Predictive Maintenance: - Anomaly detection can be used to monitor the condition of machinery and equipment such as turbines, pumps, and engines. By continuously analyzing sensor data, it can predict when maintenance is needed based on deviations from normal operating conditions. This proactive approach can prevent unexpected breakdowns and reduce downtime.

2. Quality Control: - In manufacturing, detecting anomalies in product quality is crucial. Anomaly detection algorithms can identify defects or deviations in products during the production process. This helps ensure that only high-quality products reach the market, reducing waste and production costs.

3.Structural Health Monitoring: - It can be applied to assess the structural integrity of buildings, bridges, and other infrastructure. It can identify structural anomalies that may indicate potential safety hazards, allowing for timely maintenance and repairs.

4. Aircraft Maintenance: - In aviation, anomaly detection is used to monitor the health of aircraft systems and engines. Deviations from normal performance can be indicative of potential issues that require attention, improving flight safety.

5. Energy Efficiency: - It can help identify abnormal energy consumption patterns in industrial facilities. By detecting anomalies, engineers can optimize energy usage and reduce operational costs.

6. Vibration Analysis: Anomalies in vibration patterns of rotating machinery can signal problems like misalignments, unbalanced loads, or bearing failures. Continuous monitoring and anomaly detection can prevent catastrophic failures.

7. Wear and Tear Analysis: Anomaly detection can track the wear and tear of mechanical components over time. By identifying abnormal wear patterns, engineers can schedule maintenance or replacement before critical failures occur.

8. Robotics and Automation: In robotic systems, anomaly detection can be used to identify unexpected behavior in robots or automated machinery. This helps ensure safe and efficient operation in manufacturing and industrial settings.

9. Fault Detection in Vehicles: It can be applied to detect faults or anomalies in automotive systems, such as engines, transmissions, or braking systems. This can improve vehicle safety and reduce maintenance costs.

10. Process Control: It is used to monitor and control various industrial processes. Detecting anomalies can help maintain consistent product quality and optimize process efficiency.

In these applications, data from sensors, IoT devices, and other sources are collected and analyzed using various machine learning and statistical techniques to detect anomalies. The goal is to identify deviations from expected behavior and take proactive measures to prevent costly breakdowns, ensure safety, and optimize performance in mechanical engineering systems.

2. Literature Review on Anomaly Detection

Anomaly detection is a critical area of research with applications spanning cybersecurity, industrial processes, and healthcare. Early techniques, rooted in statistical methods, focused on identifying deviations from expected norms. However, with the rise of big data, machine learning, and deep learning, novel approaches have emerged.

Machine learning-based methods, including Support Vector Machines and k-Nearest Neighbors, offer adaptability to various data types, while deep learning techniques such as Autoencoders and Variational Autoencoders excel in capturing complex patterns in high-dimensional data.

Unsupervised learning, represented by One-Class SVMs and Isolation Forests, enables anomaly detection with minimal reliance on labeled data. Online anomaly detection models have gained prominence, addressing real-time and streaming data applications.

Challenges persist, including model interpretability, robustness against adversarial attacks, and scalability to big data. Domain-specific adaptations are crucial, with anomaly detection finding applications in areas like finance, cybersecurity, and IoT security.

Future research will focus on improving the interpretability of models, enhancing their robustness, and addressing the computational demands of deep learning in big data contexts. Anomaly detection remains a vital component of data-driven decision-making, shaping the evolving landscape of data analytics and cybersecurity.

3. Methodology

A. List of different types of Classifiers to find the accuracy of the dataset.

1. LINEAR REGRESSION

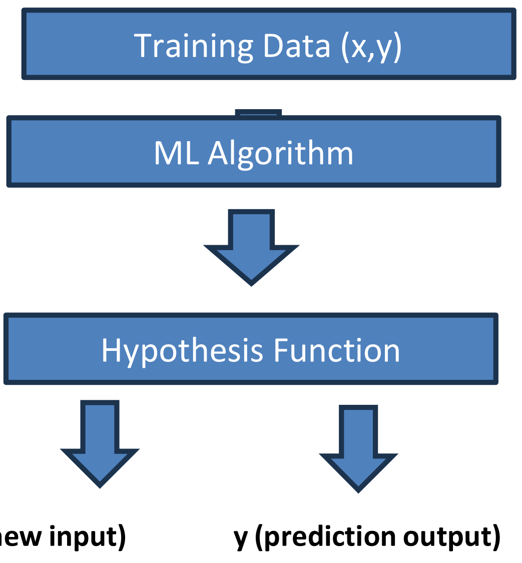
Linear regression is a supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features. Regression is where we predict the continuous output or predicting continuous numerical. In regression set of records are present with X and Y values and these values are used to learn a function so if you want to predict Y from an unknown X this learned function can be used. In regression we have to find the value of Y, So, a function is required that predicts continuous Y in the case of regression given X as independent features. Classification is where we have to detect the categories. Linear regression is typically used for predicting continuous numerical values, such as predicting a future temperature or estimating the next data point in a time series. While it's not a direct choice for anomaly detection, you can use linear regression in a way that indirectly identifies anomalies in mechanical engineering data.

Supervised Learning

⇩ ⇩

Regression | **Classification**

(Where we predict the continuous output predicting Continuous numerical) | (where we has to detect category)



The goal of the algorithm is sto find the best linear equation that can predict the value of the dependent variable based on the independent variables. The equation provides a straight line that represents the relationship between the dependent and independent variables. The slope of the line indicates how much the dependent variable changes for a unit change in the independent variable(s). The equation of linear regression is: -

$$y = wx + wo$$

- Anomalies in our data can be identified by comparing the actual values of the target variable with the predicted values from your linear regression model. Unusually large discrepancies between the predicted and actual values can be indicative of anomalies or unexpected events.

2. Logistic Regression

Logistic regression is a supervised machine learning algorithm (x, y) need both data. It is primarily used for binary classification tasks, in which only 2 categories are there where the goal is to predict whether an observation belongs to one of two classes. However, it can be adapted for anomaly detection in mechanical engineering by framing the problem as a binary classification task. The equation of a logistic regression is :-

$$\omega_0 + \omega_1 x_1 + \omega_2 x_2 = 0$$

Sigmoid Function:

The sigmoid function is a mathematical function used to map the predicted values to probabilities. It maps any real value into another value within a range of 0 and 1. o The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

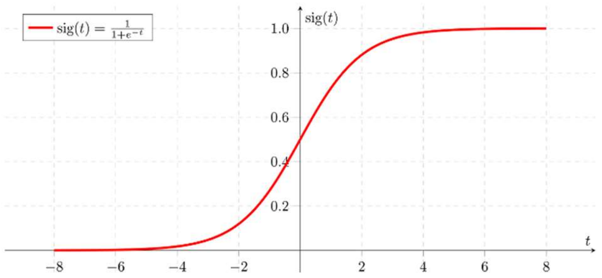


Fig. 1 : Graph of sigmoid function

- Once your model is trained, you can use it to predict whether new data points fall into the anomaly (class 1) or normal (class 0) category. If the predicted probability of belonging to class 1 is above a certain threshold, you can classify it as an anomaly.

It's important to remember that logistic regression assumes a linear relationship between the input features and the log-odds of the binary outcome. If the relationship is more complex, more advanced techniques like decision trees, random forests, or support vector machines (SVMs) might be more suitable. Additionally, logistic regression may require a substantial amount of labeled data for effective training, which can be a limitation in cases where anomalies are rare.

In summary, logistic regression can be adapted for anomaly detection in mechanical engineering by framing the problem as a binary classification task. However, it's important to consider the nature of your data and the characteristics of anomalies to determine if logistic regression is the most suitable approach or if more advanced techniques are required.

3. KNN

K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection. It is do both classification and regression. It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data.

- To detect anomalies in new data points, the KNN algorithm calculates the distance between the new point and its K-nearest neighbors in the training dataset. If the majority of the K-nearest neighbors are normal data points, the new point is classified as normal; otherwise, it's classified as an anomaly.

Despite these limitations, KNN can be a useful and interpretable tool for anomaly detection in mechanical engineering, especially when dealing with relatively small and well-labeled datasets.

4. Naive Bayes

Naive Bayes is a probabilistic classification algorithm, and while it's typically used for classification tasks, it can also be adapted for anomaly detection in mechanical engineering. It is also a supervised algorithm.

Bayes Theorem:

Bayes theorem is also known as the Bayes Rule or Bayes Law. It is used to determine the conditional probability of event A when event B has already happened. The general statement of Bayes' theorem is "The conditional probability of an event A, given the occurrence of another event B, is equal to the product of the event of B, given A and the probability of A divided by the probability of event B." i.e.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

where,

P(A) and P(B) are the probabilities of events A and B

P(A|B) is the probability of event A when event B happens

P(B|A) is the probability of event B when A happens

- Once your Naive Bayes model is trained, you can use it to predict whether new data points fall into the anomaly (class 1) or normal (class 0) category. You'll calculate the probability of each class for a new data point and classify it based on the higher probability.

Despite these limitations, Naive Bayes can be a useful and interpretable tool for anomaly detection in mechanical engineering, especially when dealing with relatively simple data structures and when the independence assumption is not too far from reality.

5. DECISION TREE

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

Decision Trees can be effectively used for anomaly detection in mechanical engineering when you have labeled data, with some examples representing normal behavior and others representing anomalies.

- Once your Decision Tree model is trained, you can use it to predict whether new data points fall into the anomaly (class 1) or normal (class 0) category. The model will traverse the tree based on feature values to make predictions.

6. K-MEANS

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on. It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

This method looks at the data points in a dataset and groups those that are similar into a predefined number K of clusters. A threshold value can be added to detect anomalies: if the distance between a data point and its nearest centroid is greater than the threshold value, then it is an anomaly.

The main difficulty resides in choosing K, since data in a time series is always changing and different values of K might be ideal at different times. Besides, in more complex scenarios where there are both local and global outliers, many outliers might pass under the radar and be assigned to a cluster.

Anomaly Detection in Network Traffic with K-means clustering

We can categorize machine learning algorithms into two main groups: **supervised learning** and **unsupervised learning.** With supervised learning algorithms, in order to predict unknown values for new data, we have to know the target value for many previously-seen examples. In contrast, unsupervised learning algorithms explore the data which has no target attribute to find some intrinsic structures in them.

Clustering is a technique for finding similar groups in data, called **clusters**. Clustering is often called an unsupervised learning task as no class values denoting an a priori grouping of the data instances are given.

In this notebook, we will use K-means, a very well-known clustering algorithm to detect anomaly network connections based on statistics about each of them. A thorough overview of K-means clustering, from a research perspective.

Goals

We expect students to:

* Learn (or revise) and understand the K-means algorithm

* Implement a simple K-means algorithm

* Use K-means to detect anomalies network connection data

Steps

1. In section 1, we will have an overview about K-means then implement a simple version of it.

2. In section 2, we build models with and without categorical features.

3. Finally, in the last section, using our models, we will detect unusual connections.

1.6 K-means

1.6.1. Introduction

Clustering is a typical and well-known type of unsupervised learning. Clustering algorithms try to find natural groupings in data. Similar data points (according to some notion of similarity) are considered in the same group. We call these groups **clusters**.

K-Means clustering is a simple and widely-used clustering algorithm. Given value of *k*, it tries to build *k* clusters from samples in the dataset. Therefore, *k* is an hyperparameter of the model. The right value of *k* is not easy to determine, as it highly depends on the data set and the way that data is featurized.

To measure the similarity between any two data points, K-means requires the definition of a distance function between data points. What is a distance? It is a value that indicates how close two data points are in their space. In particular, when data points lie in a *d*-dimensional space, the Euclidean distance is a good choice of a distance function, and is supported by MLLIB.

In K-means, a cluster is a group of points, with a representative entity called a centroid. A centroid is also a point in the data space: the center of all the points that make up the cluster. It's defined to be the arithmetic mean of the points. In general, when working with K-means, each data sample is represented in a *d*-dimensional numeric vector, for which it is easier to define an appropriate distance function. As a consequence, in some applications, the original data must be transformed into a different representation, to fit the requirements of K-means.

1.6.2. How does it work?

Given *k*, the K-means algorithm works as follows:

1. Randomly choose *k* data points (seeds) to be the initial centroids

2. Assign each data point to the **closest centroid**

3. Re-compute (update) the centroids using the current cluster memberships

4. If a convergence criterion is not met, go to step 2

We can also terminate the algorithm when it reaches an iteration budget, which yields an approximate result. From the pseudo-code of the algorithm, we can see that K-means clustering results can be sensitive to the order in which data samples in the data set are explored. A sensible practice would be to run the analysis several times, randomizing objects order; then, average the cluster centers of those runs and input the centers as initial ones for one final run of the analysis.

1.6.3. Illustrative example

One of the best ways to study an algorithm is trying implement it. In this section, we will go step by step to implement a simple K-means algorithm.

What is the Euclidean distance ?

The euclidean distance is the ordinary distance between two points in a n-dimensional space; in other words, it's the straight-line distance between the two points that translates into the rooted sum of the squared difference of the two points for each different dimension.

$$|X - Y| = \sqrt{\sum_{i=1}^{i=n} (X_i - \gamma_i)^2}$$

X = Array or vector X

Y = Array or vector Y

Xi = Values of horizontal axis in the coordinator plane

Yi = Values of vertical axis in the coordinate plane

n = Number of observations

We put a flag sqrt in order to make even faster the computation of the error since we would square root to power again later and those operations would simply cancel out.

Using numpy actually, we were able to improve the speed of this function and this will help us in the long computations in the final questions of the notebook.

The speed test was performed with simple numbers and run 10 thousand times and we can still see that numpy lowers the time taken from 15 seconds to 10 seconds (different results may appear above because those cells have been run many times after this comment has been wrote) but afterwards, when using this functions many more times, the difference is sensible.

Next, we will test our algorithm on [Fisher's Iris dataset] and plot the resulting clusters in 3D.

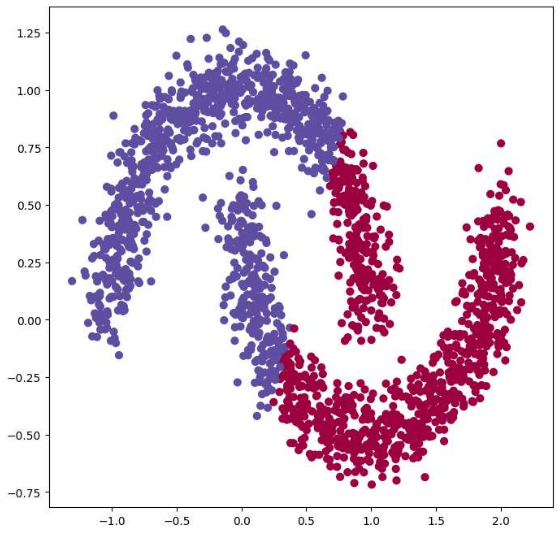


Fig.2: One of the main problems of clustering techniques.

With this dataset, we can clearly see one of the main problems of the clustering technique. The visual division of the two clusters is totally different than the effective division computed by the algorithm.

The blob dataset above shows another problem with clustering. The left plot represents the data clusterized in 3 clusters with the centroids initialized with a random seed set to 0. The right plot instead has no constraints on the initial centroids and so they are always chosen randomly.

This example shows us that the effectivness and the accuracy of the clustering technique is highly dependant on different factors and one of them is the choice of the initial centroids.

That's enough about K-means for now. In the next section, we will apply MMLIB's K-means on Spark to deal with a large data in the real usecase.

2. Usecase: Network Intrusion

Some attacks attempt to flood a computer with network traffic. In some other cases, attacks attempt to exploit flaws in networking software in order to gain unauthorized access to a computer. Detecting an exploit in an incredibly large haystack of network requests is not easy.

Some exploit behaviors follow known patterns such as scanning every port in a short of time, sending a burst of request to a port... However, the biggest threat may be the one that has never been detected and classified yet. Part of detecting potential network intrusions is detecting anomalies. These are connections that aren't known to be attacks, but, do not resemble connections that have been observed in the past.

In this K-means is used to detect anomalous network connections based on statistics about each of them.

2.1. Data

The data comes from [KDD Cup 1999]. The dataset is about 708MB and contains about 4.9M connections. For each connection, the data set contains information like the number of bytes sent, login attempts, TCP errors, and so on. Each connection is one line of CSV-formatted data, containing 38 features: back, buffer_overflow, ftp_write, guess_passwd, imap, ipsweep, land, loadmodule, multihop, neptune, nmap, normal, perl, phf, pod, portsweep, rootkit, satan, smurf, spy, teardrop, warezclient, warezmaster.

Many features take on the value 0 or 1, indicating the presence or absence of a behavior such as `su_attempted` in the 15th column. Some features are counts, like `num_file_creations` in the 17th columns. Some others are the number of sent and received bytes.

2.2. Clustering without using categorical features

First, we need to import some packages that are used in this notebook.

2.2.1. Loading data

There are two types of features: numerical features and categorical features.

Currently, to get familiar with the data and the problem, we only use numerical features. In our data, we also have pre-defined groups for each connection, which we can use later as our "ground truth" for verifying our results.

Note 1: we don't use the labels in the training phase!!!

Note2: in general, since clustering is un-supervised, you don't have access to ground truth. For this reason, several metrics to judge the quality of clustering have been devised. For a short overview of such metrics. Note that computing such metrics, that is trying to assess the quality of your clustering results, is as computationally intensive as computing the clustering itself!

Write function `parseLine` to construct a tuple of `(label, vector)` for each connection, extract the data that contains only the data points (without label), then print the number of connections.

Where,

* `label` is the pre-defined label of each connection

* `vector` is a numpy array that contains values of all features, but the label and the categorial features at index `1,2,3` of each connection. Each `vector` is a data point.

For this initial part, we are not considering the categorical features in indexes 1,2 and 3 that are:

1. protocol type
2. service
3. flag

We have approx. 5 million connections. Let's see how the data is divided.

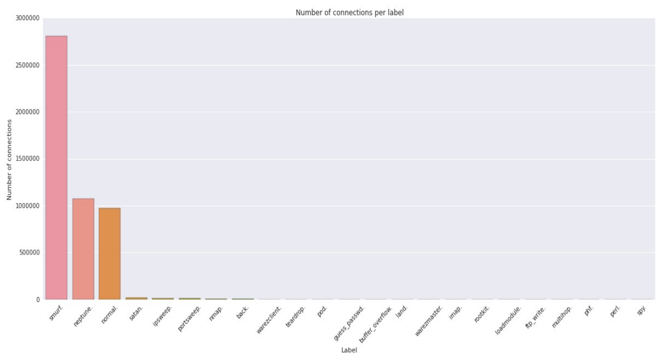


Fig.3: Data is not equally divided.

There are 23 different labels but actually we can immediately see that the data is not equally divided.

Of the nearly 5 million connections, more than a half are labelled as *smurf* and other 2 millions are equally divided as *neptune* and *normal.* We computed above the amount of connections that this first 3 labels amount for.

As we can see, of the 4898431, 4852684 are dedicated to just 3 labels leaving less than 40 thousand connections to the remaining 20 labels.

This initial look inside the data may hint us on how the data is divided in the space: since so much data has the same label, probably it will be packed up together and leave all the other labels as "outliers".

Using K-means algorithm of MLLIB, cluster the connections into two groups then plot the result. Why two groups? In this case, we are just warming up, we're testing things around, so "two groups" has no particular meaning.

You can use the following parameters:

1. `maxIterations=10`
2. `runs=10`
3. `initializationMode="random"`

Discuss the result from your figure.

Each connection has 38 attributes. If we want to cluster the data we have to decide which are the best features to use on the axis to plot the data. For this first example, let's simply use the first 3 features to do a 3D plot.

Let's first see how the data is divided.

In our tests, we got completely random results; the first time we run the training, we got the 5 million connections divided almost equally; the second time we got all connections but 5 in one cluster and 5 connections of label "portsweep" in the second cluster; while the third time we have all connections in one cluster and none in the second one.

This is an evidence of two things:

1. Clustering main problem of centroid initializations
2. Curse of dimensionality that can be the explanation of the random results we get; the majority of the data is stuck together and goes inside the same cluster.

We'll try now to plot the data.

First of all we get the values of the 3 axis, and then we scatter the plots. We will not plot all of the 4 million connections but only a subset of those.

As of now, we will just use the first 3 attributes as the first axis.

Also the plot above doesn't make a lot of sense but that is just the case because we plotted the first values on 3 random axis.

A relevant information that we can get from this plot, even though it plots only the first 1000 connections data, is that feature 2 is almost the same (other than the point on the right side) for every point and so maybe this feature doesn't contain much information.

A good technique to get the best features and to understand how much of the information is contained in them is to use PCA.

From Principal Component Analysis

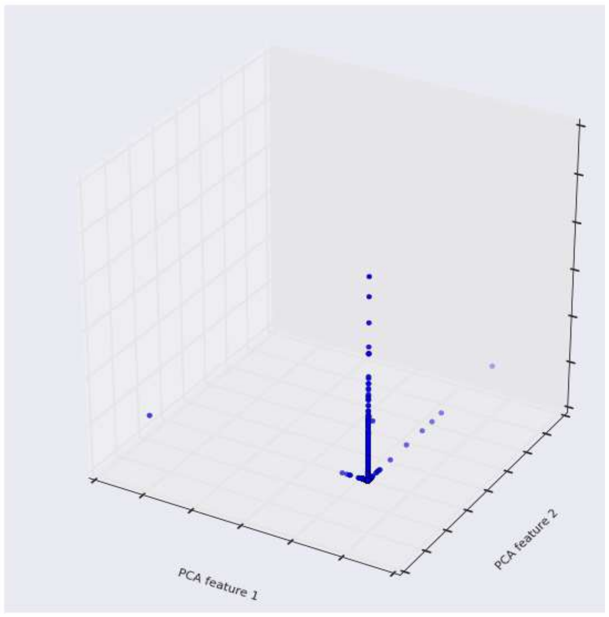


Fig.4: Principal Component Analysis(a)

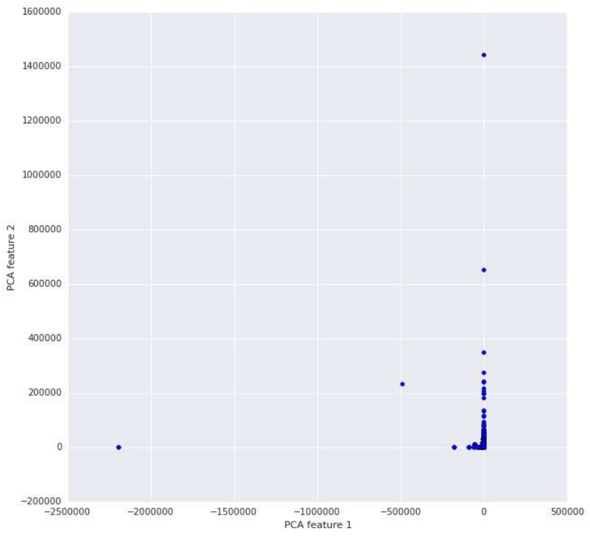


Fig.4: Principal Component Analysis (b)

Above are the plots of the Principal Component Analysis did with k=3 and k=2.

First of all, what is PCA? It is a procedure to transform a set of variables, possibly correlated, into another set of uncorrelated variables called principal components. The number of principal components is less or equal to the number of initial variables and this technique is really useful when we want to reduce dimensionality. If we start for example with 38 dimensions, after the transformation we get a set of principal components and, according to the number of dimensions we want, we get the first k principal components. And why do we get the first and not random ones? Because the transformation is defined in such a way that the principal components are ordered with respect to how much variability of the data they account for.

Looking at the first plot, the one where we used 3 principal components, we can see that the majority of the data is packed up at the intersection of the 3 axis and that the data lies especially on the z-axis. This means that, even though we used 3 principal components, 2 of them don't have really high variance and in fact they don't show too much data.

The same happens for the second plot where we use just two components, and the data is still packed together in $(0,0)$ (in the next plots we'll try to zoom in).

But why all the data is perpendicular to each other?

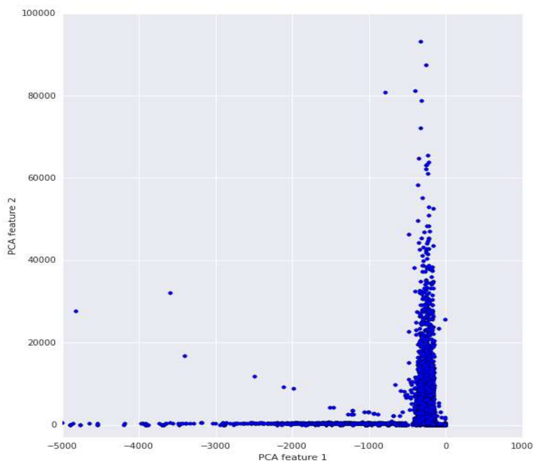
Because, looking at the plot of the data (not with PCA), we can see that the points are already almost all perpendicular to each other and so, given that the eigenvalues are uncorrelated (and so orthogonal to each other), when multiplied with the points themselves, they get projected on just one of the axis every time.

We used the PCA function in the MLLib library otherwise it would have been impossibile to compute it in the full dataset.

Actually, when we initially tried to use PCA on a small sample of data and we managed to use the PCA function in the sklearn library, we had the possibility to understand the amount of variance with the explained_variance_ratio function and we discovered that with just two principal components, almost 100 percent of the information was represented.

This is a proof of what we saw before analyzing the data: that since so much of the data is represented with just 3 labels, it will be all packed together.

Looking closer at the second plot, we see that the dimensions are huge and so let's try to zoom into $(0,0)$.



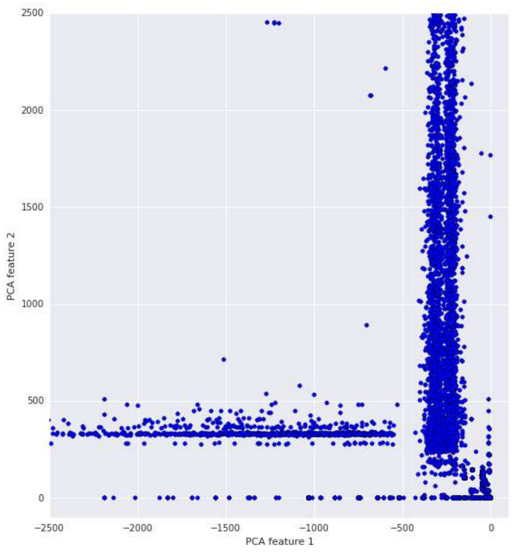


Fig.5: Plot shows better the effectiveness of the PCA.

This plot shows better the effectiveness of the PCA. All of the data is compressed in the previous plot because there are some points with a much higher value in those dimensions that make all the data shrink together. Zooming in, we can see that the majority of the points span in a small contour of the dimensions.

UPDATE

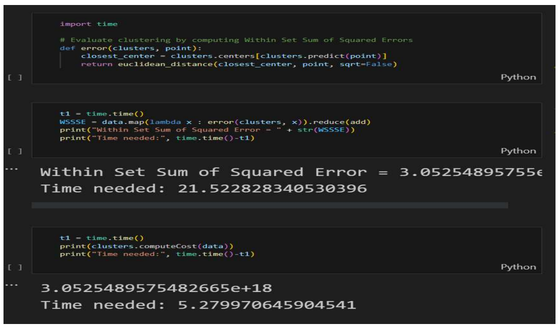
We discovered that using a different machine learning library, more specifically pyspark.ml.feature instead of pyspark.mllib.feature, the pca is more complete.

We used this library just to provide the variance explanation since we had no time to change the code of the plotting.

As printed above, we can see that the first two principal components provide the totality of the information and the third component adds 0% of information.

This can also be seen in the plots above where one of the dimensions in the 3D-plot is useless.

2.2.3 Evaluating model



```
import time

# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(clusters, point):
    closest_center = clusters.centers[clusters.predict(point)]
    return euclidean_distance(closest_center, point, sqrt=False)
```

```
t1 = time.time()
WSSSE = data.map(lambda x : error(clusters, x)).reduce(add)
print("Within Set Sum of Squared Error = " + str(WSSSE))
print("Time needed:", time.time()-t1)
```

```
Within Set Sum of Squared Error = 3.05254895755e
Time needed: 21.522828340530396
```

```
t1 = time.time()
print(clusters.computeCost(data))
print("Time needed:", time.time()-t1)
```

```
3.0525489575482665e+18
Time needed: 5.279970645904541
```

Fig.6: Code Implementation

In the KMeans documentation, we found a method called public double computeCost

(RDD&lt;Vector&gt; data)that actually computer the same information but in a much faster way.

We wrote the time of both methods and the improving factor is approximately 10.

As we get from the implementation, all of the data but 5 elements goes in one cluster while 5 elements labeled as *portsweep* go in the second cluster. (at least that was the result at the time of the comment but sometimes as we said before the result are different).

One cluster usually gets almost all the classes while the other one gets 1 or 0 classes (portsweep, more specifically).

The clusters are heavily unbalanced due to the disparity of the whole dataset and this tells us one important thing: that the majority of the point is stuck together and there is a really small amount of outliers (only 5 were recognized). There are surely more than 5 outliers but they hide for sure inside the first cluster.

The choice of k=2 could be better adapted; since there are 23 classes, it would make much more sense to use k=23 or more.

2.2.4. Choosing K

How many clusters are appropriate for a dataset? In particular, for our own dataset, it's clear that there are 23 distinct behavior patterns in the data, so it seems that k could be at least 23, or likely, even more. In other cases, we even don't have any information about the number of patterns at all (remember, generally your data is not labelled!). Our task now is finding a good value of $k$. For doing that, we have to build and evaluate models with different values of $k$. A clustering could be considered good if each data point were near to its closest centroid. One of the ways to evaluate a model is calculating the Mean of Squared Errors of all data points.

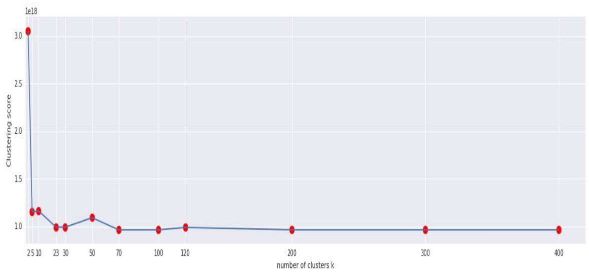


Fig.7: Plot of different increasing number of clusters.

We tried a different increasing number of clusters and we plotted the results above.

As we can see from the plot and more importantly, as we know from the theory, increasing the number of clusters improves the clustering score. And this is quite obvious because if we put a huge number of clusters, for example the same as the number of points in the dataset, each point will be in its own cluster and the distance from the center will be zero. This is why simply increasing k to improve our clustering is not an efficient and sensible choice and we should find a compromise between number of clusters and clustering score.

The plot shows us that the initial choice of k=2 was absolutely nonsense being that the clustering score is really high with respect to the rest of the tests. Afterwards, with k=5, we have a huge drop when we test the KMeans. From k=23 onwards, the clustering score doesn't decrease anymore in a consistent way and so going further than 10/23 clusters doesn't seem to be a good choice.

This actually makes sense if we think about the data we are clustering: we have 23 different labels but actually many of them are pretty similiar and with similiar attributes and so when we use less clusters, they still stay packed together and increasing the clusters will just make more clusters really near to each other not improving the clustering score.

Other than simply looking at the plot, another more methodological solution could have been to apply the Minimum Description Length Principle that tries to find the best optimization for k.

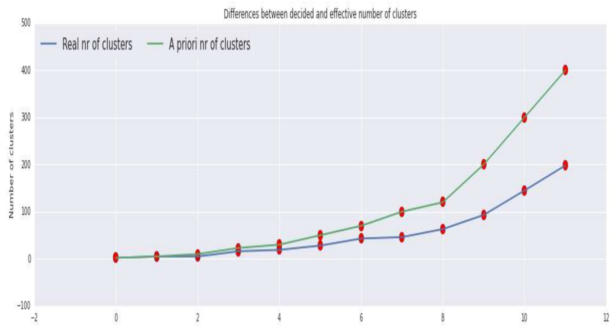


Fig.8: Plot shows clustering Score stayed same after a certain point.

We tried to go further in details on the previous plot and, knowing from theory that increasing the number of clusters usually improves the error, we were wondering why the clusteringScore stayed the same after a certain point.

In the plot above, we decided to plot the number of clusters per run, k, together with the effective number of clusters. ***What do we mean with effective number of clusters?*** When the kmeans algorithm runs, the various points change cluster as the centers get updated. If, after reassigning the points, one of the cluster becomes empty, it is deleted from the list of clusters.

As we can see from the plot above, when the number of clusters is small, none of them is deleted but, as we increase the number of clusters, the effective number of clusters doesn't increase with the same pace and this is why the clusteringScore doesn't improve a lot. The trend is that after 20 clusters, approximately half of the desired clusters gets deleted. We can see this from the slope and also from the detailed numbers in the printing above that:

- 30 drops to 11
- 50 -> 26
- 70 -> 38
- 100 -> 48
- 120 -> 60
- ...
- 400 -> 190

2.2.5 Normalizing features

K-means clustering treats equally all dimensions/directions of the space and therefore tends to produce more or less spherical (rather than elongated) clusters. In this situation, leaving variances uneven is equivalent to putting more weight on variables with smaller variance, so clusters will tend to be separated along variables with greater variance.

In our notebook, since Euclidean distance is used, the clusters will be influenced strongly by the magnitudes of the variables, especially by outliers. Normalizing will remove this bias.

Each feature can be normalized by converting it to a standard score. This means subtracting the mean of the feature’s values from each value, and dividing by the standard deviation

$$nomalize\ i = \frac{feature\ i - \mu i}{\sigma i}$$

Where,

$normalize\ i$ is the normalized value of feature i

$\mu\ i$ is the mean of feature i

σi is the standard deviation of feature i

What did we do thanks to normalization? Essentially, the shape remains the same and the only thing that changes is the value for each attribute for each connection. We did this in order to equalize the variance for each dimension.

We actually implemented our own function that uses as argument labelsAndData and not just data because we will need it later for question 12.

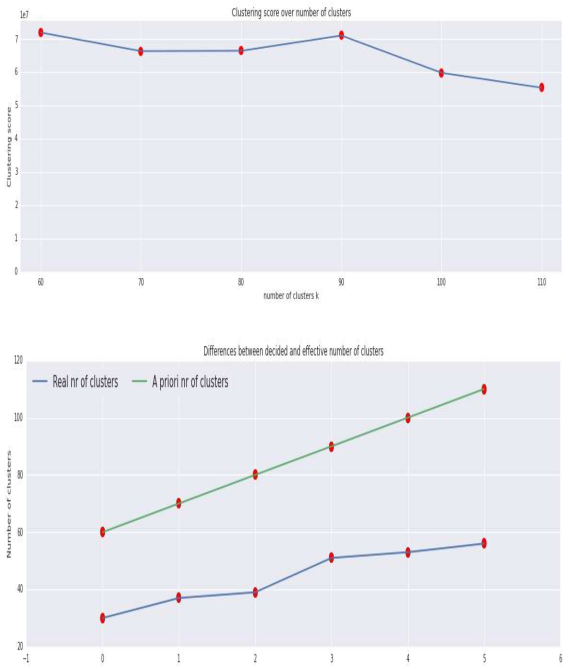


Fig.9:

First of all, before commenting the plot, we notice one important fact:

The scale for the WSSSE dropped consistently from a magnitude of 10^{18} to a magnitude of 10^{7}

Other than this, we can still see the trend that increasing the number of clusters reduces the clustering score although here is much less visible since the number of clusters is really high.

Also, plotting the real and the expected number of clusters, we can see that the trend is always similiar and that the effective clusters are always much smaller (almost half) of the expected ones.

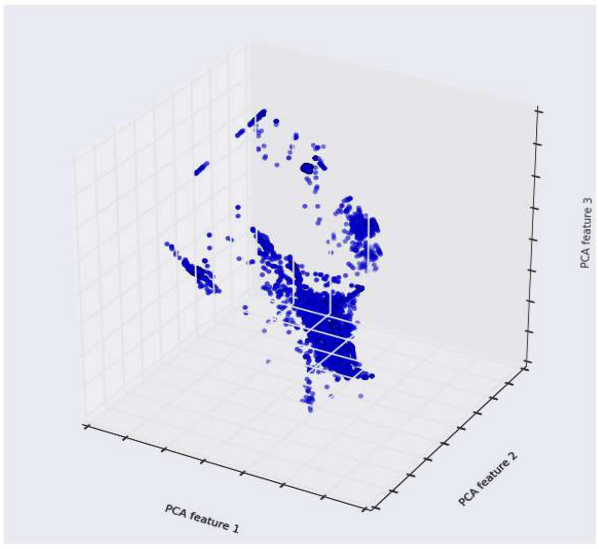


Fig.10: Normalizing values allowed a more standard variance for the different attributes and we can see the improvements in both plots.

The first plot is done with the first three features and this is to show that even if we normalize the data, and so the values change, the relationship between different points remain the same in fact, if you have a look at the first plot in question 3, it didn't change at all (without considering the colors that are random for each class and that the data is sampled so some points may differ, but the distribution is mantained).

The second plot is more interesting and shows a sample of 1% of the data plotted on the first three principal components.

If we look side by side at this plot and at the PCA plot in question 3, we can see that thanks to data normalization, the data is spread more equally on the different principal components. We can see that the data is not packed together like it was before and so this may improve our anomaly detection thanks to a better division of clusters.

2.3. Clustering using categorical features

2.3.1 Loading data

In the previous section, we ignored the categorical features of our data: this is not a good idea, since these categorical features can be important in providing useful information for clustering.

The problem is that K-means (or at least, the one we have developed and the one we use from MLLib) only work with data points in a metric space. Informally, this means that operations such as addition, subtraction and computing the mean of data points are trivial and well defined. For a more formal definition of what a metric space is.

What we will do next is to transform each categorical feature into one or more numerical features. This approach is very widespread: imagine for example you wanted to use K-means to cluster text data. Then, the idea is to transform text data in *d*-dimensional vectors, and a nice way to do it is to use .

There are two approaches:

Approach 1: mapping **one** categorical feature to **one** numerical feature. The values in each categorical feature are encoded into unique numbers of the new numerical feature. For example, ['VERY HOT','HOT', 'COOL', 'COLD', 'VERY COLD'] will be encoded into [0,1,2,3,4,5]. However, by using this method, we implicit assume that the value of 'VERY HOT' is smaller than 'HOT'... This is not generally true.

Approach 2: mapping **one** categorical feature to **multiple** numerical features. Basically, a single variable with *n* observations and *d* distinct values, to *i* binary variables with n observations each. Each observation indicating the presence (1) or absence (0) of the dth binary variable. For example, ['house', 'car', 'tooth', 'car'] becomes

[

[1,0,0,0],

[0,1,0,0],

[0,0,1,0],

[0,0,0,1],

]

We call the second approach "one-hot encoding". By using this approach, we keep the same role for all values of categorical features.

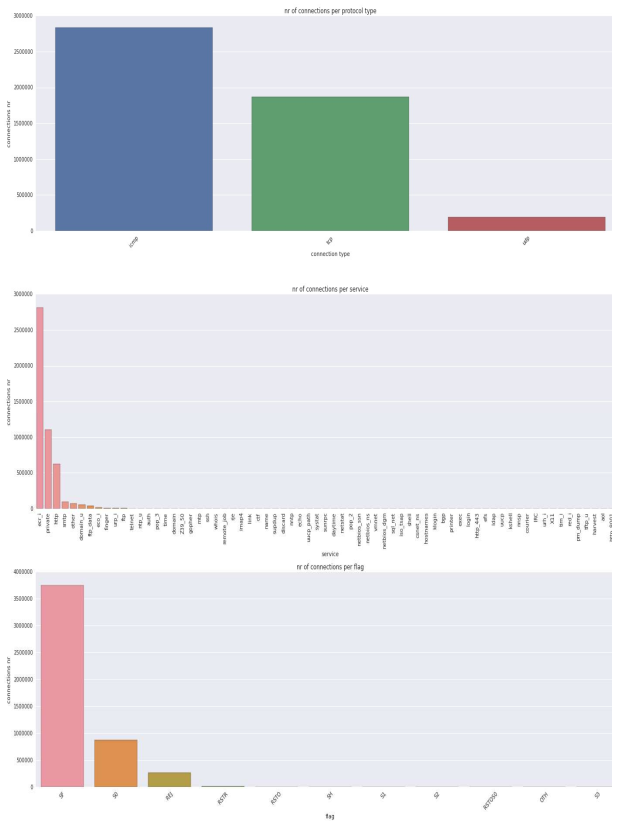


Fig.11: After plotting the data and see how this categorical features split the data.

We are trying to take account of all the different attributes we have at our disposal. Initially we didn't care about the categorical attributes (index 1,2,3) but now, with Hot Encoding, we are trying to see if the clustering with also this attributes becomes better.

Before using this new, more complete, data, let's see how this features separate our data set of nearly 5 million elements.

The first attribute describes the connection type and can take 3 values, tcp, udp and icmp. The data is almost equally divided between icmp and tcp, leaving less than 50 thousand connections to udp.

The second attribute describes the service and the third attribute describes the flags. As we can see, unfortunately, both this attributes concentrate much of the data in just 3 or 4 labels out of the 70, for the second attribute, and 11, for the third attribute.

We wrote *unfortunately* because having a not-so-distributed division of the data make us think that the clustering will not improve that much leaving the data as packed as before.

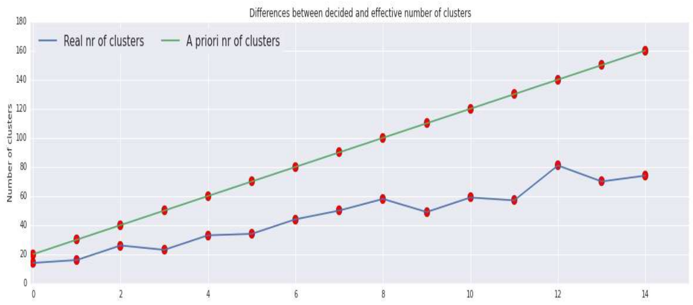


Fig.12

In our opinion, checking the clsutering score for just clusters with 80+ clusters up to 160 didn't make a lot of sense. We want to see also how it performs with a smaller *k*, but still enough large to contain all the possible labels. This is why we increased the k_vector from 20 up to 160.

As before, we plotted both the trend of the clustering score and the relationship between effective and desired number of clusters.

As we can see, the clustering score improves as *k* increases, but remains almost the same from k=100 onwards.

We can also see a first "bump" when k increases from 80 to 90 that is the best improvement we were able to get from one hop to the next one.

We'll try to perform the next tests, anomaly detection and silhouette with k = 90, a good compromise between number of clusters and clusteringScore.

2.4. Anomaly detection

When we have a new connection data (e.g., one that we never saw before), we simply find the closest cluster for it, and use this information as a proxy to indicate whether the data point is anomalous or not. A simple approach to decide when there is an anomaly or not, amounts to measuring the new data point’s distance to its nearest centroid. If this distance exceeds some thresholds, it is anomalous.

Even here we can see that even though we start with 120 clusters, the model degenerates to only 68 clusters since some of them remain with 0 elements.

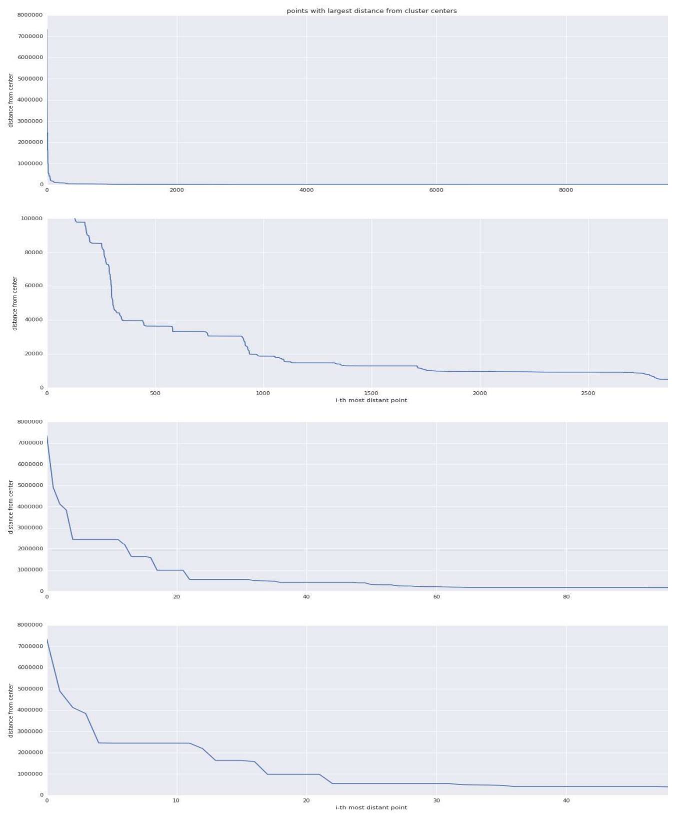


Fig.13:

The first plot shows the 10000 points with the largest distance from the cluster. As we can see, after an initial peak of 30 (probably the anomalous connections), the distance from the cluster tends to get smaller. This means that the majority of the clusters are packed up together and the outliers could be easily detected.

The second plot is a zoom in of just the bottom part (with a limit on the y range) and we can see that actually zooming in, the plot continues to slope down until 2400-2500 connections and then stays to a "small" distance for the rest of the plot.

The next two plots are just a zoom in on the top-100 and top-50 furthest connections.

Other than the first plot, this zoom-in allow us to decide which is the best threshold to use to divide anomalous connections from the normal ones.

As we can see, the first 20/30 points span a distance from 1 million to 7 million, and considering the first 2500 points (remember that the full dataset is 5 million points so 2500 points is just 0.05%) the distance drops from 7 million to almost 5000. This number alone doesn't say a lot, since to evaluate the error the distance is squared, but noticing that after that slope the distance is almost constant makes us think to put that number as threshold.

We can try to plot the nr of anomalous connections per threshold to have a better idea:

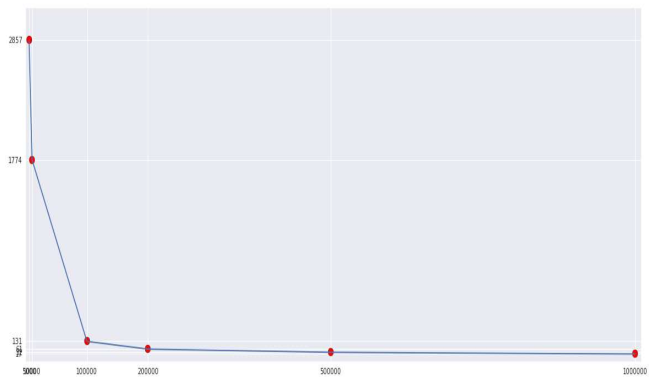


Fig.14:

This plot shows the number of connections we would label as anomalous if we would select a certain threshold. As obvious, increasing the threshold will determine less anomalous connections.

An important thing to know is that for a company that has to automatically label anomalous connections for, let's say, further checks, it's better to "grab" and then check all true anomalous connections together with some non-anomalous connections than leave unchecked some true anomalous connections.

This problem is well known and states that it's not a good metric to see the accuracy of a model because, taking this case as an example, since the anomalous connections would be relatively small, let's say 0.5%, if we would state that everything is non-anomalous we would still get an accuracy of 99.5% *but* the accuracy would be high just thanks to the ratio between anomalous and non-anomalous connections.

To begin with, we would start putting as a threshold 5000 and so evaluating 2323 connections as anomalous (the numbers will change when we run again the notebook).

Let's try to print this connections and see what are their labels.

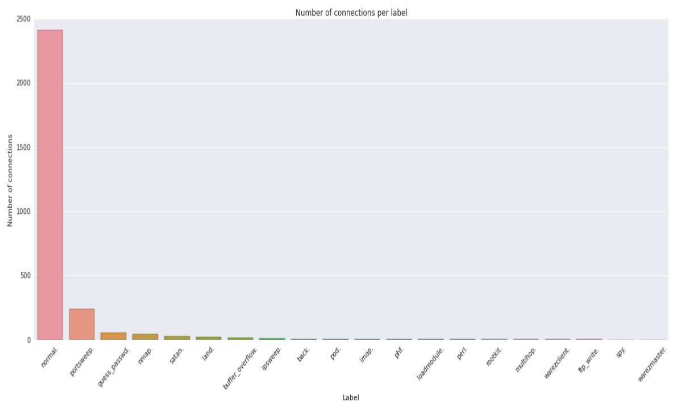


Fig.15:

In the plot above, we chose as a threshold 5000 to plot the labels of the 2857 anomalous connections we would detect.

It's interesting to see that of the almost 3 thousand anomalous connections, almost 2500 are labelled as normal.

Actually, we expected a result similiar to this one. ***Why?*** Because all the other labels are *known* attacks and in our opinion an anomaly is a connection different to all the other ones already seen like, for example, a new type of attack, an unknown error, a strange traffic... In our opinion, a guess_passwd attack or a buffer_overflow attack is known and traffic analyst know how to recognize it instead, a model like this one should be used to allow them to recognize *new* type of dangers. And this is why, since this should be *unknown* dangers that the system doesn't know how to label, in our opinion are all labelled as normal connection.

In our opinion a model like this one would be used in parallel with another model able to predict which type of connections are happening; let's say a decision tree that uses all the attributes to label the connections and recognize known attacks, and this model here to recognize new attacks and strange traffics.

Let's try to increase the threshold to see if, even with a smaller number of anomalous connections detected, the normal label would be the major one.

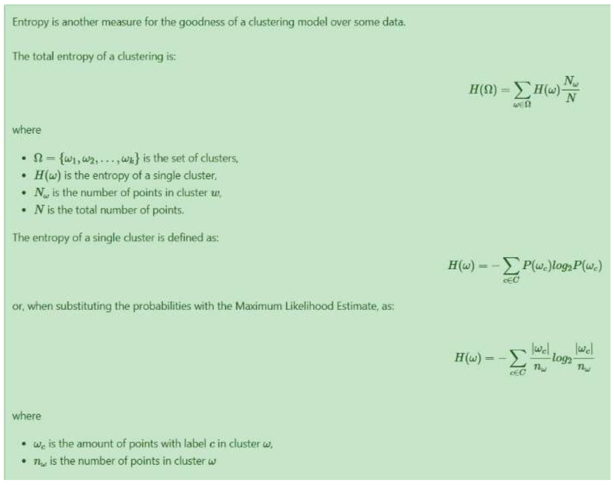
We increased the threshold up to a point where only 131 anomalies would be detected.

As we can see, the <code>normal</code> label is still the most present one, giving a hint that our reasoning in the comment before might be correct.

ENTROPY

Entropy is another measure for the goodness of a clustering model over some data.

The total entropy of a clustering is:



Entropy is another measure for the goodness of a clustering model over some data.

The total entropy of a clustering is:

$$H(\Omega) = \sum_{\omega \in \Omega} H(\omega) \frac{N_\omega}{N}$$

where

- $\Omega = \{\omega_1, \omega_2, \dots, \omega_k\}$ is the set of clusters,
- $H(\omega)$ is the entropy of a single cluster,
- N_ω is the number of points in cluster w,
- N is the total number of points.

The entropy of a single cluster is defined as:

$$H(\omega) = -\sum_{c \in C} P(\omega_c) log_2 P(\omega_c)$$

or, when substituting the probabilities with the Maximum Likelihood Estimate, as:

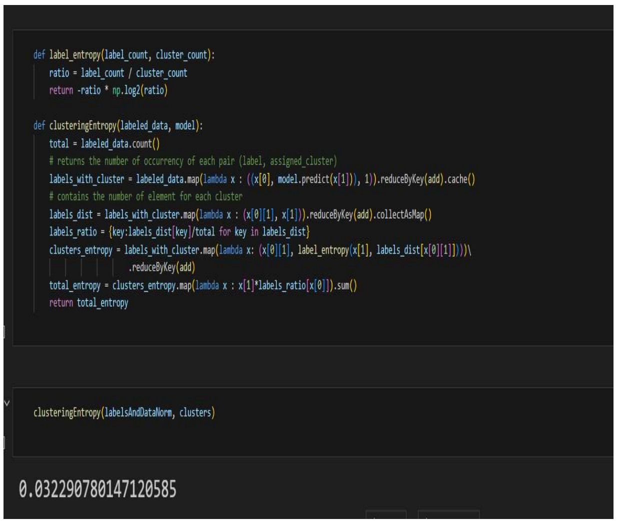
$$H(\omega) = -\sum_{c \in C} \frac{|\omega_c|}{n_\omega} log_2 \frac{|\omega_c|}{n_\omega}$$

where

- ω_c is the amount of points with label c in cluster ω,
- n_ω is the number of points in cluster ω

Fig.16: Entropy

In simple words, entropy represents the "mess" or "disorder" inside each cluster. And how disorded is defined? As the number of connections of different labels, and so different connections, that were grouped together. In other words, when we cluster, we don't use the label attribute but just the other 38 attributes. To evaluate the entropy, we look at the labels and we check if the different labels were grouped together or are apart from each other. This is a good metric since each label should represent a good grouping of the different connections.



```
def label_entropy(label_count, cluster_count):
    ratio = label_count / cluster_count
    return -ratio * np.log2(ratio)

def clusteringEntropy(labeled_data, model):
    total = labeled_data.count()
    # returns the number of occurrency of each pair (label, assigned_cluster)
    labels_with_cluster = labeled_data.map(lambda x : ((x[0], model.predict(x[1])), 1)).reduceByKey(add).cache()
    # contains the number of element for each cluster
    labels_dist = labels_with_cluster.map(lambda x : (x[0][1], x[1])).reduceByKey(add).collectAsMap()
    labels_ratio = {key:labels_dist[key]/total for key in labels_dist}
    clusters_entropy = labels_with_cluster.map(lambda x: (x[0][1], label_entropy(x[1], labels_dist[x[0][1]])))\
                        .reduceByKey(add)
    total_entropy = clusters_entropy.map(lambda x : x[1]*labels_ratio[x[0]]).sum()
    return total_entropy
```

```
clusteringEntropy(labelsAndDataNorm, clusters)
```

```
0.032290780147120585
```

Fig.17: Code Implementation

The entropy for the chosen *K* seems really good. It means that the data is divided in distant and well grouped clusters. It actually makes sense since even if we started with $k=90$, the effective number of clusters is 39, not much bigger than the number of labels and so it should be able to group the different data in a correct way.

Let's try to plot the silhouette to get a second analysis on the clusters.

SILHOUETTE

Silhouette analysis can be used to study the separation distance between the resulting clusters.

The silhouette for each point is evaluated as:

where $a(i)$ represents the similiarity of point $i$ to its own cluster and $b(i)$ represents the dissimiliarity with respect to the neighbouring cluster (the second nearest). We can define $a(i)$ as how well the point is assigned to its cluster and $b(i)$ as how well it would have been assigned to the neighbour cluster.

In our evaluations, $a(i)$ represents the distance of the point i to the center of its own cluster instead $b(i)$ represents the distance to the center of the neighbouring cluster or, in other words, the distance to the second nearest center.

The complex and complete definition of $a(i)$ and $b(i)$ actually tell to evaluate the distance with respect to all points of the cluster (own cluster for $a(i)$ or different cluster for $b(i)$ and then average with the number of points. This definition would have a complexity of $\mathcal{O}(n^2)$ and with the dimensions of our dataset it would be too time consuming.

With our definition, considering only the centroids and not all the points of the cluster, the complexity drops to $\mathcal{O}(kn)$ where k is the number of centroids that is much much smaller than the dataset.

We used this definition thinking about physics: when we have to compute the force on a rigid body, instead of evaluating it over all points of the body, we just use the center of mass. In the same way, here, we use the centroids to represent the clusters.

Because of the definition of the k-means algorithm, we know for sure that $a(i)$ is smaller than $b(i)$ (otherwise point i wouldn't have been assigned to its cluster, and so the formula can be rewritten as:

$$s = \frac{b - a}{max(a, b)}$$

The silhouette is a value between -1 and +1 but in our case it's always a positive value between 0 and +1. If the value is high, near +1, it means that the point is close to its own center and distant from the neighbouring cluster instead if it's close to 0, it means that the point is nearly in the middle between two clusters and so the prediction is not so good.

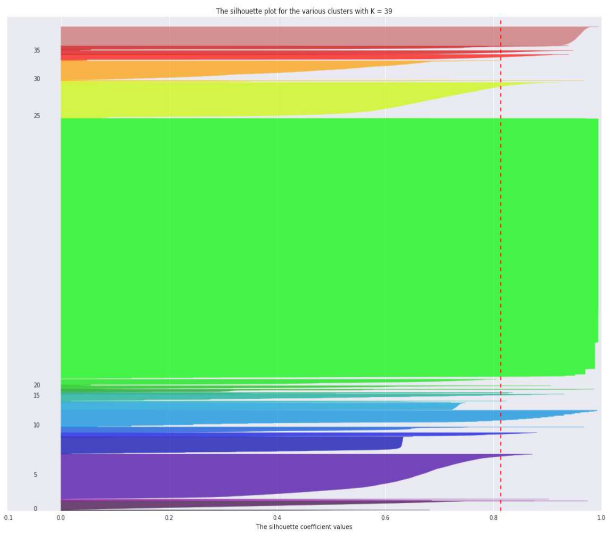


Fig.18: Silhouette plot

The plot above is a visual explanation of the goodness of the cluster and uses the silhouette value to help reasoning on the model.

The red dotted vertical line represents the silhouette_avg.

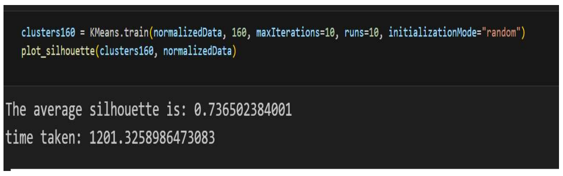
Every coloured curve instead represents the silhouette of each point inside a cluster (each with its own color).

A good model, should have a high average and have all clusters' silhouette to be almost at the average level.

We actually have a good average but this is due to the large cluster that has a really high silhouette and so shifts the whole average. Unfortunately, many other clusters fall behind.

This may be explained by the fact that the majority of the data is packed together, as we saw in the initial questions and so falls all in one cluster (probably the big one).

We'll try now to plot the silhouette of a model with a higher k, such as 160.



```
clusters160 = KMeans.train(normalizedData, 160, maxIterations=10, runs=10, initializationMode="random")
plot_silhouette(clusters160, normalizedData)
```

```
The average silhouette is: 0.736502384001
time taken: 1201.3258986473083
```

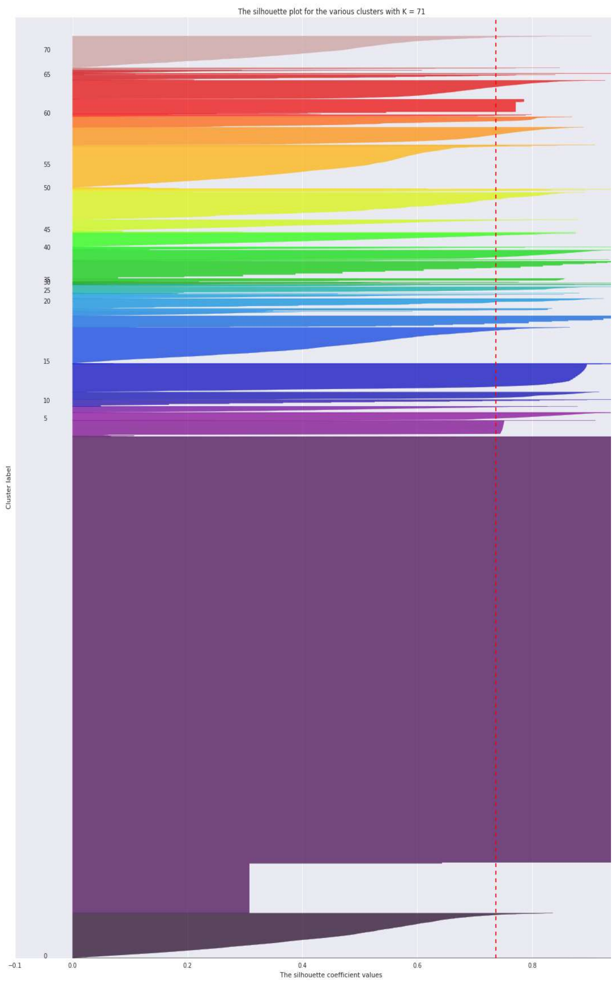


Fig.19:

As we can see above, even with a higher k, there is still a huge cluster containing the majority of the points.

In this case, perhaps, the average is a little bit lower and the other clusters all manage to mantain a higher average.

Without considering the large cluster, it seems that all the other clusters would be equally divided.

In the end, we could say that the clustering is good, and the huge cluster is almost inevitable with this dataset.

This algorithm is not performing so well but at least it's working. It returns the value of the k centroids.

To predict a cluster for a point, it is sufficient to run find_closest_centroid with parameters the point itself and the array of the centroids which will return the index of the closest centroid.

4. Conclusion

In conclusion, anomaly detection is a pivotal field with wide-ranging applications in our data-driven world. From traditional statistical methods to cutting-edge machine learning and deep learning techniques, the evolution of anomaly detection has been marked by remarkable progress and increasing complexity.

While these methods have shown tremendous promise in identifying deviations from expected patterns, several challenges persist. The need for model interpretability, robustness against adversarial attacks, and scalability to handle massive datasets remains at the forefront of research and development efforts.

Moreover, the domain-specific nature of anomaly detection cannot be overlooked. Its applicability spans across diverse sectors, from finance to healthcare, each presenting unique challenges and opportunities for customization.

Looking ahead, future research will undoubtedly focus on addressing these challenges and refining anomaly detection techniques to meet the evolving needs of cybersecurity, industrial processes, and emerging domains such as IoT security. As data continues to play a central role in decision-making, anomaly detection will remain an indispensable tool for safeguarding systems and ensuring data integrity in an increasingly interconnected world.

5. References

Dataset link:

https://en.m.wikipedia.org/wiki/Iris_flower_data_set

Code implementation:
https://drive.google.com/drive/folders/1RPdOXlDImUY8Q490rVbuDM9YdUv1tghJ?usp=sharing

https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/anomaly-detection

https://chat.openai.com/?model=text-davinci-002-render-sha

https://chat.openai.com/c/215367bc-e3db-4361-bb89-f059fc46dbb8

https://www.geeksforgeeks.org/naive-bayes-classifiers/

http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

http://erikerlandson.github.io/blog/2016/08/03/x-medoids-using-minimum-description-length-to-identify-the-k-in-k-medoids/

http://en.wikipedia.org/wiki/Iris_flower_data_set

Our definition of entropy and our code below follow the definitions from the https://nlp.stanford.edu/IRbook/

https://nlp.stanford.edu/IRbook/html/htmledition/evaluation-of-clustering-1.html

The code for ploting the silhouette is inspired from

http://scikitlearn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html