Task 8: Kmean Evaluation of KDD99 Dataset

CANEVET Gaspard, PARISSE Jeanne, SPATENEDER Andreas, MEYER Sven

June 10, 2021

Contents

1 Introduction

This task consists of an evaluation of anomaly detection system made of Kmeans algorithm with KDD99 dataset. An anomaly detection system is a method, algorithm that allows machine, human to detect any behaviour that look suspicious. Our system knows what is normal behaviour and thus can detect abnormal behaviour. In that case we are going to use Kmeans algorithm and KDD99 training dataset to create a model that is going to gather data in clusters. Those clusters describe normal or abnormal behaviour. If a new data point is too far from normal clusters we will be able to raise an alert.

In the first part we are going to dive deeper in KDD99 dataset. This is crucial to fully understand our dataset in a Datascience studies. After designing our dataset, we will design our Kmeans algorithm. We have to find the proper number of cluster K for our study. Thus we will use the elbow curve method. Finally, we will train, test and evaluate our model with several metrics.

Our algorithms can be found on a github repository [4].

2 KDD99 Dataset

We load our dataset from the website [1]. We decided to use the 10% dataset to save time computing. This dataset is made of 494021 rows and 42 columns. Each row is a connection record from a LAN that simulates a typical U.S. Air Force LAN. Columns are information regarding the connection. The 41 columns can be separated in 3 families: basic, content and traffic features and the last one is the label of the connection. There are four main connection categories: normal, DOS attacks, R2L attacks, U2R attacks and probing attacks. You can find more precise information about dataset rows and columns on the page [2].

We first look at missing values in the dataset but apparently, there is no missing value. It saves us lot of time and prevents us to think about a method to fill in missing values.
Then we have to deal with non continuous variables. Indeed Kmeans algorithm only work with continuous variables. This is why we are going to drop non continuous variable from our dataset. It could be interesting to think about a method that could take discrete variable in account like hierarchical clustering with mahalanobis distance.

After deleting columns we have 494021 rows and 32 columns. Before starting the study for number of clusters K we have to center our data and split them into a training set and a test set. The test set is made of 30% of the initial data set and the training set is made of the leftover 70%. It is also sufficient to take a look to data distribution in three datasets. For instance if the train dataset is full of "abnormal" data, our model will learn how to see differences within abnormal data but not within abnormal and normal data.

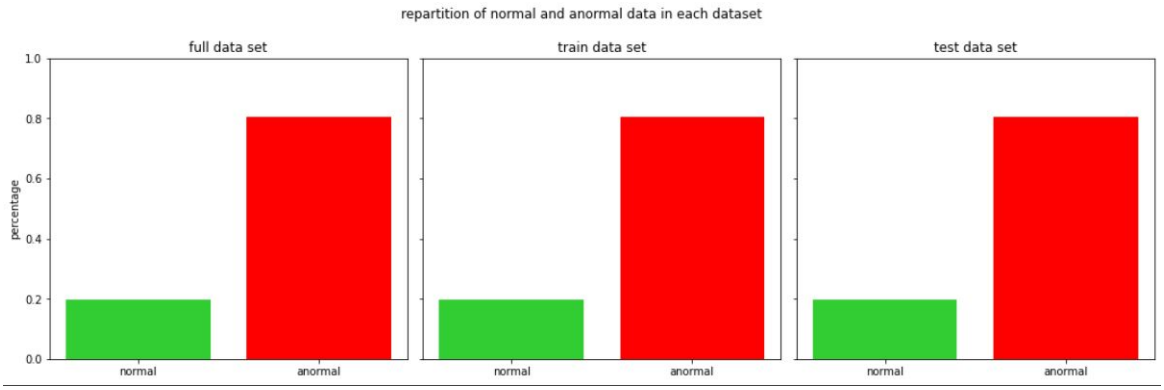

Figure 1: data distribution in each dataset

We can see there that data distribution is the same in each data. There is roughly 20% of normal data and 80% of abnormal data. It is important to have the same repartition in

each dataset but the predominance of abnormal behaviour is not necessarily a good thing. As a reminder, the goal of an anomaly detection system is to learn patterns of normal behaviour in order to detect abnormal behaviour. Here our algorithm could learn patterns that highlight abnormal behaviour rather than patterns that highlight normal behaviour. In that case, our model will be an intrusion detection system instead of anomaly detection. We have to be aware of that. Even if we have good results with our test dataset at the end, it could be interesting to test our model with quite new intrusion behaviour. If we had bad results, it would mean that we have overfitting on abnormal behaviour. Our work on KDD99 dataset can be found in the part data exploration in our JupyterNotebook.

We are now going to study the number K of clusters we have to look for in our Kmean algorithm. It is a crucial hyperparameter. We have to choose it carefully as the effectiveness of our model depends on it.

3 Number of clusters K

Let me remind you what we have to perform our Kmeans algorithm : Two centered dataset (be careful to centered training and test set on their own to prevent data leakage), one for the training part and one for the test part, respectively made of 70% and 30% of the initial dataset. We decided to only keep the continuous variables because we are evaluating distances between datapoints with euclidean distance.

We are now going to perform Kmeans algorithm on our training dataset with a growing number of clusters ($K = 15, ..., 49$) to determine the best number of cluster K* using elbow curve methodology and sklearn library. We could also have use silhouette method to find the best number of cluster K* but it takes to much time to be computed with such a dataset size and our computer performance. We have simply not been able to perform one silhouette study.

Let start our elbow curve study. First we need to give some parameters to our sklearn Kmeans function:

- dataset -> training set
- Number of clusters K -> between 15 and 49
- n_init -> Kmeans algorithm depends on the initial random centroids. This variable represents the number of time we are going to run a Kmean algorithm with differents initial centroids. We decided to set it to 15 for the training phase

Then after completing the Kmeans algorithm, we are interesting in the .inetia_ output. This variables is the sum of squared distances of samples to their closest cluster center [3]. Our goal is to minimize this variable and to find a turning point as in the picture below.

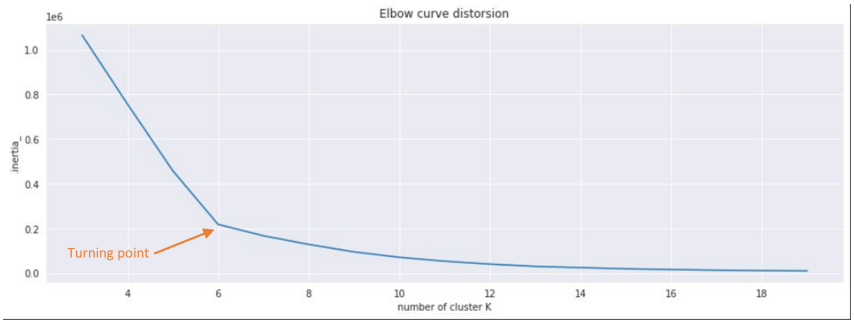

Figure 2: typical elbow curve

Here we can see that K=6 is the best number of cluster according to elbow curve method. Nevertheless, it could also be interesting to take a look to cluster distribution to validate the number of cluster K.

After running our Kmeans algorithm like we explain before we obtained the elbow curve below.

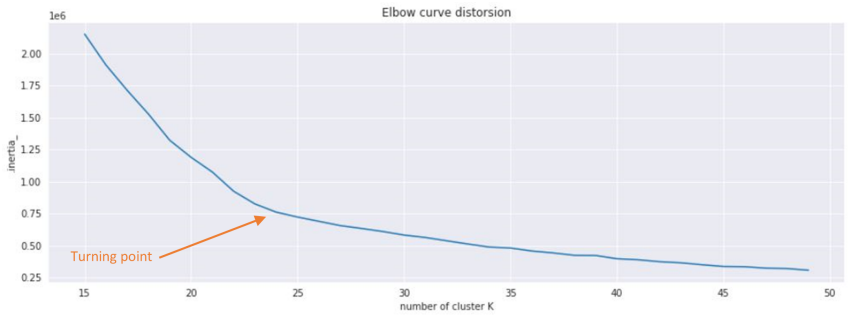

Figure 3: elbow curve with k=[15,..,49] and n_init=15

Here, the turning point is not as strong as in the Figure 2 but there is still one. It seems that K*=24 is the proper number of clusters for our study according to the elbow curve method. Nevertheless, we are going to dive into data distribution along each clusters. It will ensure that our parameters K* is correct.

Now we have our number of clusters K*. We are going to train our Kmeans model. Thus we can have a look to data distribution along each cluster. The settings for the Kmeans algorithm are:

- dataset -> training set
- Number of clusters K -> 24
- n_init -> 30. We take a bigger nto limit the impact of initial centroids seed.

After having trained our model we use their data labels to gather some information about data distribution. We decided to compute the percentage of abnormal and normal behaviour and the percentage of data in the cluster. there is, in the appendix, a picture of the table that summarize our cluster data but below we can see a bar plot that brings to the light computed variables.

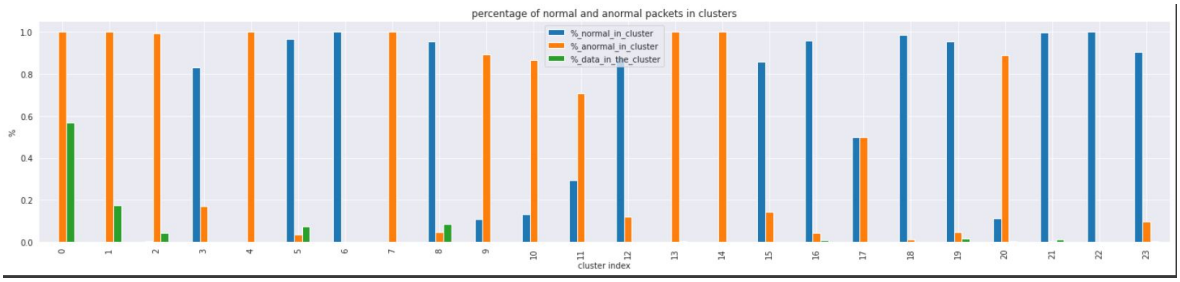

Figure 4: percentage of abnormal and normal behaviour and the percentage of data in each cluster

Here, we can see that most of the clusters are made of predominant type of data (abnormal or normal behaviour) especially the one with important amount of data. For instance the three first clusters that contain more than 70% of the total dataset are full of abnormal behaviour. Now, the tricky part of the study will be smallest clusters like the number 11. It is made of 30% of normal data and contains 297 data point. We could definitely raise these questions:

- Can we assign this clusters to a certain type of behaviour?
- Do we really care about small clusters or can we overlook them?

We will try to answer to these question in the coming part but first let summarize the part on the definition of the proper number of clusters K*. First we plotted the elbow curve and we saw that **K*=24** was the turning point. Finally we checked that the data distribution along each cluster were sufficient and it seems to be.

4 Evaluation of the model

In that part we are going to evaluate our models using our test dataset and some metrics. First we have to assign each cluster to a certain type of behaviour:

- Conforming to security policy
- Security violation

We used a quite trivial algorithm to define clusters. We compare the proportion of normal and abnormal data in each cluster and use the biggest one as the cluster definition. In case of equality we decided to define the cluster as abnormal. Indeed false negative data are far more dangerous than false positive. In the first case it could wreak havoc an entire network and in the second case it could only waste some time of the system administrator. Then we used our trained model to predict test datapoint belonging. Finally we computed the confusion matrix and obtained the result below:

	Conforming to security policy	Security violation
anomaly	1193	117734
normal	29124	156

Table 1: Confusion matrix

It seems that our model is performing well. Let use some metrics to highlight it. We are going to use Three metrics:

$$precision = \frac{Truepositive}{Truepositive + Falsepositive} \tag{1}$$

$$recall = \frac{Truepositive}{Truepositive + Falsenegative} \tag{2}$$

$$F1 - score = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} \tag{3}$$

The first metric, the *precision*, emphasizes the impact of False positive. For instance, in the case of machine learning model that aims to advise people to bet on a football match, false positive is a predicament. Indeed, people can loose money if the model shared wrong advice. On the other hand if the model said that the game is too dangerous to bet on, people do not bet and cannot loose money even if the match was one sided at the end.

The second metric, the *recall*, highlights the impact of false positive. For example, in the case of machine learning model that aims to detect potential tumor with MRI pictures, false positive is a predicament. Indeed, people could dodge more accurate tumor detection test and it could lead to terrible outcomes. On the other hand if the model said that the picture is suspicious, the patient could go through more accurate test and be categorized not ill at the end.

Finally, the last metric, the *F1-score*, is a trade off between the two previous metrics. It is an interesting metrics for project were both false positive and false negative are predicaments. In our case both false positive and false negative have to be limited. false positives are a waste of time for system administrators and false negatives could wreak havoc an entire network. It is obvious that false negatives are far more dangerous but false positive must not be underestimate. This is why we decided two compute the three metrics.

	test dataset
precision	0.9987
recall	0.9899
F1-score	0.9943

Table 2: metrics computed with the test dataset

Apparently we have really good results as all three metrics are above 98%. For a machine learning model that is quite impressive. Nevertheless, this result have to be put into perspective. This dataset is made of old data from 1999 network traffic. It is known that intrusion techniques are changing really fast and it seems that our model is quite strong to detect old intrusion but it could be interesting to test it with more current data. On top of that, training and test dataset could be quite similar and it could have a sort of data leakage.

To analyze the result a little further, we can calculate the False Positive rate to see how many false alarms we are going to get and how many system administrator is needed to manage those. We assume that the results we have are what happens during a day.

$$FalsePositiveRate = \frac{FalsePositive}{FalsePositive + TrueNegative} \tag{4}$$

We get a result of 0.005328, so 0.5%.

During task 5, we said that a system administrator needs on average five minutes to investigate an alarm. Keeping that in mind, we can calculate how many false alarms there is for 5 millions data, which represents 7 weeks of data. 0.5% of 5 millions is 25 000 false alarms, so we will need 125 000 minutes/2083 hours to take care of those false alarms during 7 weeks.

In task 5, we also determined that one system administrator must spend at most 1 hour per day on false alarms, which means that during 7 weeks, one system administrator will spend 7x7 = 49 hours on false alarms. So, in conclusion, we will need $2083 \div 49 = 42.5$, so 43 system administrators to take care of those false alarms.

5 Conclusion

The evaluation of KDD99 through K-means algorithm provide a very good overview of what is clustering and what are detection system. When applying the elbow method, we found that 24 is the ideal number of clusters this data set. The evaluation of the model seems to corroborate this as our precision, recall and F1-score are all above 98%, which is quite an impressive result in a machine learning model.

However, the whole process is not perfect. KDD99 is a 1999 data set, and data set tend to be outdated very quickly. It is very unlikely that a data set older than 20 years is still relevant for today's problems. Furthermore, the data set has a huge number of anomalies compared to normal data. An anomaly detection system has to learn patterns of normal behaviour in order to detect abnormal behaviour. Hear, the algorithm is going to learn more patterns that are linked to abnormal behaviour instead of normal behaviour. It is not assured that the model will react well in front of new intrusion behaviour. Also, with more up to date data, the results could be worse than what we conclude. In conclusion, the analysis of this data set is an excellent exercise to understand how clustering and k-means work, but its relevance and direct application to real-world problems is limited due to the aforementioned problems.

6 Appendix

	Nb_data_in_cluster	Nb_normal_in_cluster	Nb_anormal_in_cluster	%_normal_in_cluster	%_anormal_in_cluster	%_data_in_the_cluster
0	196439	22.0	196417.0	0.000112	0.999888	0.568048
1	60649	9.0	60640.0	0.000148	0.999852	0.175380
2	14998	80.0	14918.0	0.005334	0.994666	0.043370
3	41	34.0	7.0	0.829268	0.170732	0.000119
4	681	0.0	681.0	0.000000	1.000000	0.001969
5	25863	24996.0	867.0	0.966477	0.033523	0.074789
6	1	1.0	0.0	1.000000	0.000000	0.000003
7	45	0.0	45.0	0.000000	1.000000	0.000130
8	29682	28325.0	1357.0	0.954282	0.045718	0.085832
9	28	3.0	25.0	0.107143	0.892857	0.000081
10	15	2.0	13.0	0.133333	0.866667	0.000043
11	297	87.0	210.0	0.292929	0.707071	0.000859
12	511	449.0	62.0	0.878669	0.121331	0.001478
13	1133	0.0	1133.0	0.000000	1.000000	0.003276
14	3	0.0	3.0	0.000000	1.000000	0.000009
15	7	6.0	1.0	0.857143	0.142857	0.000020
16	3125	2992.0	133.0	0.957440	0.042560	0.009037
17	2	1.0	1.0	0.500000	0.500000	0.000006
18	305	301.0	4.0	0.986885	0.013115	0.000882
19	5854	5592.0	262.0	0.955244	0.044756	0.016928
20	996	111.0	885.0	0.111446	0.888554	0.002880
21	3625	3617.0	8.0	0.997793	0.002207	0.010483
22	2	2.0	0.0	1.000000	0.000000	0.000006
23	1512	1368.0	144.0	0.904762	0.095238	0.004372

Figure 5: cluster summarize

7 References

[1]$http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html$
[2]$http://kdd.ics.uci.edu/databases/kddcup99/task.html$
[3]$https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html$
[4]$https://github.com/CANEVETGASPARD/KDD-99_{d}ataset_{s}tudy$