Alma Mater Studiorum – University of Bologna

Discriminately Boosted Clustering

Artificial Intelligence in Industry (2021-2022)

Author : Zarmina Ursino - [zarmina.ursino@studio.unibo.it](mailto:zarmina.ursino@studio.unibo.it)

Source code available : <https://github.com/Zarmina97/DBC_Ai_industry>

Summary

[Abstract 3](#_Toc107164764)

[Introduction 3](#_Toc107164765)

[Data exploration 3](#_Toc107164766)

[Convolutional Autoencoders -intro 4](#_Toc107164767)

[Data preparation 4](#_Toc107164768)

[Autoencoder Model 4](#_Toc107164769)

[DBC 5](#_Toc107164770)

[Results 6](#_Toc107164771)

[PCA plot and silhouette plot 6](#_Toc107164772)

[Evaluation metrics 7](#_Toc107164773)

[Conclusions 7](#_Toc107164774)

[References 8](#_Toc107164775)

# Abstract

Deep clustering is a new research direction that combines deep learning and clustering. It performs feature representation and cluster assignments simultaneously, and its clustering performance is significantly superior to traditional clustering algorithms. We observe that existing deep clustering algorithms either do not well take advantage of convolutional neural networks or do not considerably preserve the local structure of data generating distribution in the learned feature space. To address this issue, I propose a deep convolutional embedded clustering algorithm in this paper. Specifically, I develop a convolutional autoencoders structure to learn embedded features in an end-to-end way.

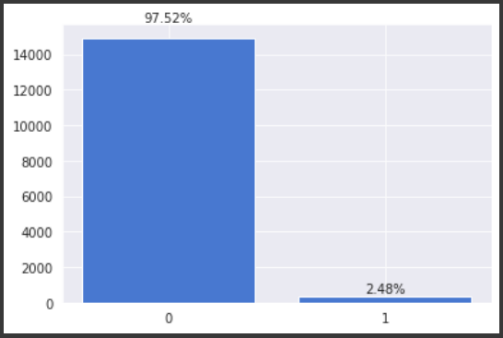
# Introduction

The dataset was provided to perform DBC. The MARCONI100 computing system installed at Cineca in early 2020 is the largest supercomputer available in Academic sector in Italy and in Europe today. It is powered by IBM Power9 processors and NVIDIA Volta V100 GPUs, employing dual-rail Mellanox EDR InfiniBand as the system network. The data was collected with a tool called Examon and the dataset is composed of several folders, a folder for each selected node.

## Data exploration

The information monitored on Marconi100's nodes is varied, ranging from the load of the different cores, to the temperature of the room where the nodes are located, the speed of the fans, details on memory accesses in writing/reading, etc.The sampling rate of the data at the source varies between 5 and 10 seconds.

However, in the data set the data are aggregated in 15-minutes intervals; in particular, the mean value ("avg: <metric\_name>") and variance ("var: <metric\_name>") are computed over each 15-minute interval. In the CSVs, each row corresponds to a different timestamp (first column on the left), therefore separated by intervals of 5 minutes.

The column called "New Label" column indicates the presence or absence of a failure on the. After loading the data, I drop the ‘label’ and ‘timestamp’ columns from the dataset. I check for any missing data to treat the missing values. There are no missing values in the dataset. I split the dataset into X and y for further model training and evaluation of the model. Here we can observe that there are two values in y. ‘0’ means normal state of the node, ‘2’ means anomalous state. So, I map all ‘2’ values with ‘1’ value for our convenience in evaluation part.

We can see that class 0 has 97,52% of entries and class 1 has only 2.48% as It’s possible to see in Figure 1. There is lot of imbalanceness in the dataset; but these class 1 values are Anomalous state. We can expect that there are very rare events. K-means is sensitive to the scale of feature values because it uses Euclidean distance as similarity metrics. For this reason, I scale these features using Minmax scaler.

Figure 1

# Convolutional Autoencoders -intro

## Data preparation

The input array passed to the CNN should be a 4D array, with a shape of *(batch\_size, height, width, depth)*, where the first dimension represents the batch size of the image and the other three dimensions represent dimensions of the image which are height, width, and depth. To do this I used the NumPy function expand\_dims(), to expand the shape. Afterwards I splitted X and target y into X\_train, X\_test, y\_train and y\_test by applying the sklearn method ‘train\_test\_split()’, with a size of the test set of 20%.

Moreover, since there is no batch size value in the input\_shape argument, we could go with any batch size while fitting the data; thus, I set input array as *Input(shape=(460, 1, 1))*

## Autoencoder Model

Immagine che contiene testo

Descrizione generata automaticamenteThe framework contains two parts. It’s possible to see in See Figure 2 a glance of the overall framework and in Figure 3 the Algorithm applied.

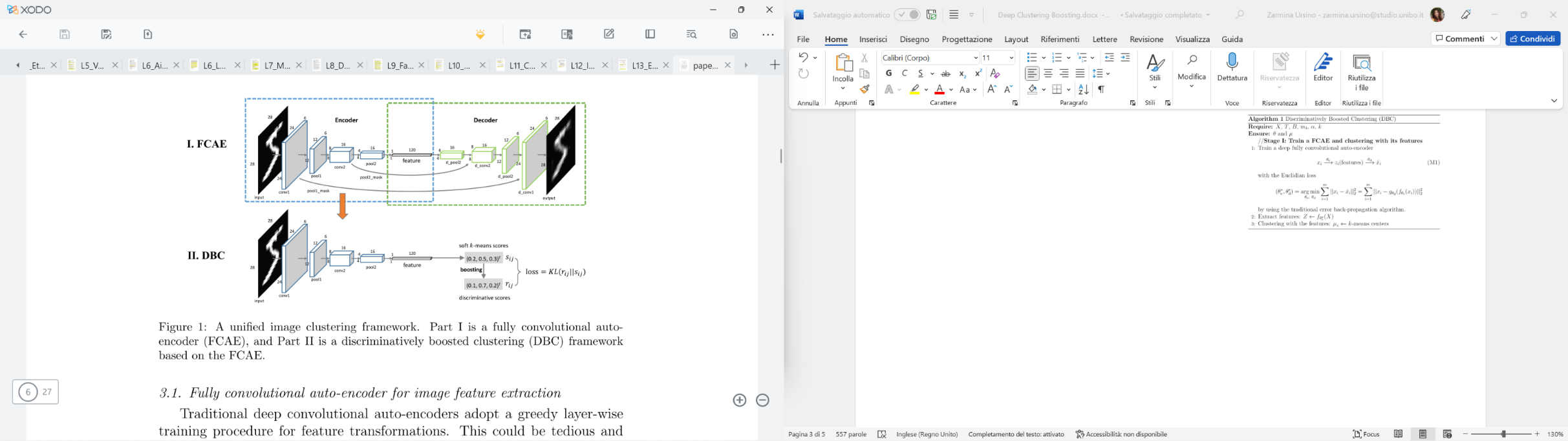


Figure 2

Figure 3

First I implement a fully convolutional auto-encoder (FCAE) which is composed of convolution-type layers (convolution and de-convolution layers) and pool-type layers (pooling and un-pooling layers). By adding batch normalization (BN) layers to each of the convolution-type layers, we can train the FCAE in an end-to-end way. This avoids a tedious and time-consuming layer-wise pretraining stage adopted in the traditional stacked (convolutional) autoencoders. To the best of our knowledge, this is the first attempt to learn a deep auto-encoder in an end-to-end manner.

The optimizer used is *‘adam*’ and the type of loss is ‘*binary\_crossentropy*’

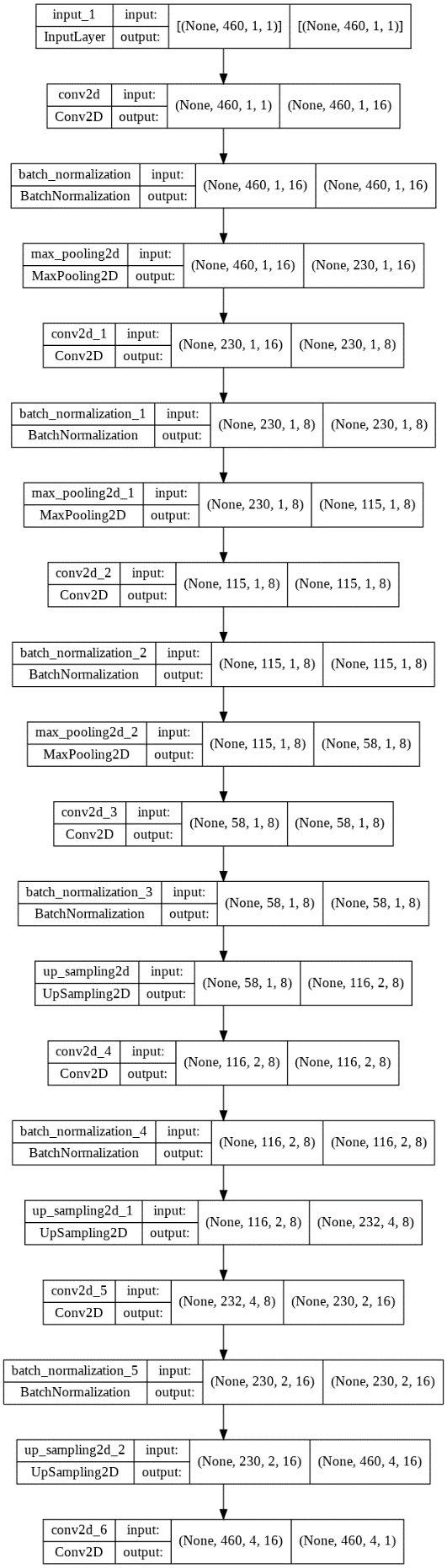


Figure 4

## 

## DBC

I propose a discriminatively boosted clustering (DBC) framework based on the learned FCAE and an additional soft k-means model. We train the DBC model in a self-paced learning procedure, where deep representations of raw images and cluster assignments are jointly learned. This overcomes the separation issue of the traditional clustering methods that use features directly learned from auto-encoders.

Immagine che contiene tavolo

Descrizione generata automaticamente

Figure 5

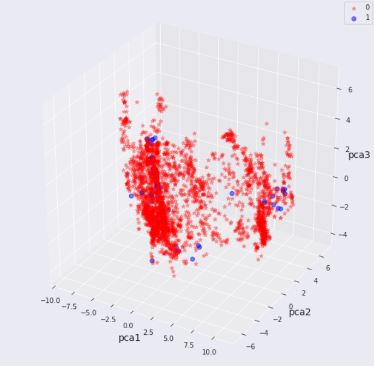
# Results

## PCA plot and silhouette plot

To visualize the clusters, I used the Principal Component Analysis (PCA), to reduce the number of features in our data set we deployed PCA (Principal Component Analysis) which tries to find the best possible subspace. It transforms our initial features into so-called components. These components are basically new variables, derived from the original ones, and they are usually displayed in order of importance.

As you can see in Figure 6, I choose 2 components while preserving as much of the original information as possible. We incorporate the newly obtained PCA scores in the K-means algorithm. In this manner we can perform segmentation based on principal components scores instead of the original features. We add the names of the segments to the labels. To visualize our clusters on a 2D visualization we choose the two components and use them as axes with the help of matplotlib and seaborn library. Thanks to PCA we are sure that the first two components explain more variance than the others.

I did the same by considering 3 components and I visualize the clusters on a 3D visualization, as you can see in Figure 7.



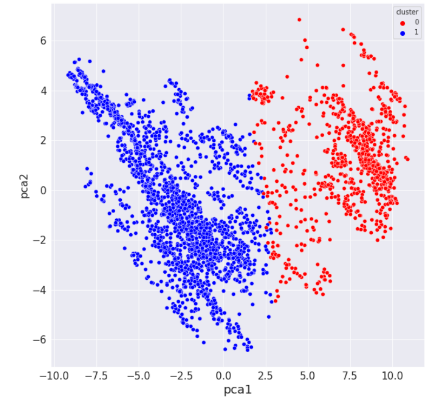
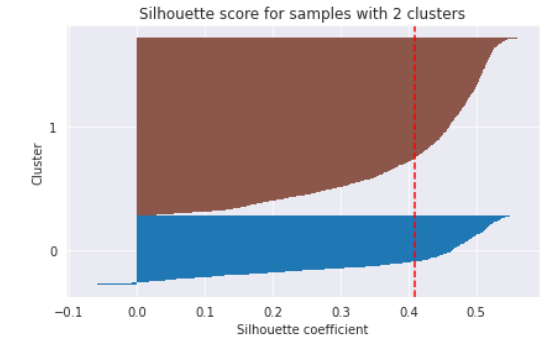


Figure 6

Figure 7



In the Silhouette plot in Figure 9 we can see that the data instance is close to the center of the cluster and instances possessing the silhouette scores close to 0 are on the border between two clusters. .

Figure 9

## Evaluation metrics

Two standard metrics are used to evaluate the experiment results explained as follows:

* Accuracy (ACC) . Given the ground truth labels {ci|1 ≤ i ≤ m} and the predicted assignments {ˆci|1 ≤ i ≤ m}, ACC measures the average accuracy:

(formula)

where g ranges over all possible one-to-one mappings between the labels of the predicted clusters and the ground truth labels

* Normalized mutual information (NMI) . From the information theory point of view NMI can be interpreted as

(formula)

where H(c) is the entropy of c and NMI(ˆc, c) is the mutual information of ˆc and c.

* Silhouette Score

Considering the hyperparameters epochs=100, the batch size= 256 and the validation size= 64, I obtained the following results.

* Accuracy = 71.913782
* Silhouette Score = 0.414561
* NMI = 0.001876

# Conclusions

# References

* Deep Clustering - [Link](https://deepnotes.io/deep-clustering)
* Convolutional Autoencoders for Image Noise Reduction - [Link](https://towardsdatascience.com/convolutional-autoencoders-for-image-noise-reduction-32fce9fc1763)
* Deep Clustering with Convolutional Autoencoders - [Link](https://xifengguo.github.io/papers/ICONIP17-DCEC.pdf)