POLITECNICO
MILANO 1863

Quality Data Analysis

(Prof. Bianca Maria Colosimo)

TEAM 7°

PROJECT WORK TITLE

Authors:

Taeha Kang – taeha.kang@mail.polimi.it

Estefany Arroyo – estefanydaniela.arroyo@mail.polimi.it

Marco Garioni – marco.garioni@mail.polimi.it

Davide Zanuso – davide.zanuso@mail.polimi.it

GIT : https://github.com/kdh7575070/polimi_22_QDA

Academic Year 2021-2022

Introduction

- For the project, 3D printed polymer PC fan cover parts were analyzed. Some parts were defect-free, while others were defective. Each part was labeled from 1 to 50, they were put one by one in the middle of the tray and passed through an in-line inspection. We could automatically acquire one grayscale real-time picture of each part when they passed below the camera. There were labels assigned for each part which were either 0 (no assignable cause), or 1 (assignable cause).

- The main objectives of the project were to design a statistical process monitoring method for the in-line detection of defects by minimizing the false alarms, to describe the performances in terms of false positive and false negative errors, to clearly state all assumptions and to justify all the choices, and to test and compare more than one approach to find the best approach.

- The structure of our project is mainly divided into 3 steps. **Data preprocessing – Main solution using PCA – Comparison with the other ML/DL algorithm**. Different types of input data were tested to pursue the best outcome, and different types of models (Regression model and control chart) were used for analyzing PCA results. Lastly, for the ML/DL algorithm, according to our research, the Random forest model and the ALEXNET model were chosen to acquire the best possible accuracy.

- Python (for the most part of the project) and Minitab (for making the control chart) are the main program used for the project. Since we understand the python programming method can be unusual, careful comments are written for each code line in every detail.

Assumptions

- Input data will have a huge impact on the outcome.

Proposed method

STEP1 – DATA PREPROCESSING [0_preprocessing.py]

- There are 4 steps made for automating this process using python, mainly **OpenCV*** package:

1. Roughly crop the input images (1280*1024 -> 400*400)
2. (2-1) Finding reference line for rotation and (2-2) rotate the images
3. Transform each pixel into binary using threshold algorithm (white 255 / black 0)
4. Precisely re-crop the images (400*400 -> 310*310)

(- For the further explanation, the way we automatically rotate images is that (1) to find one extended line of side of each square object using Canny Edge detection algorithm and Hough line algorithm from OpenCV, (2) to calculate the angle between the line and the central axis, and (3) to rotate at the amount of the opposite value of the angle)

- The main reason we used python programming is that to prove the assumption of this paper, we should efficiently have a control on some of the input parameters (e.g. crop size and position, binary transformation threshold). For the modeling part, we tried using both binary data (process order of 1-2-3-4) and greyscale data (process order of 1-2-4) to compare the results.

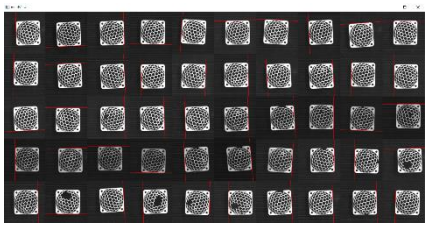

Figure 1: after process 2-1

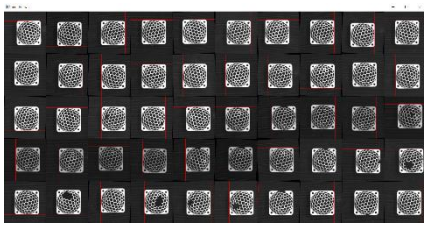

Figure 2: after process 2-2

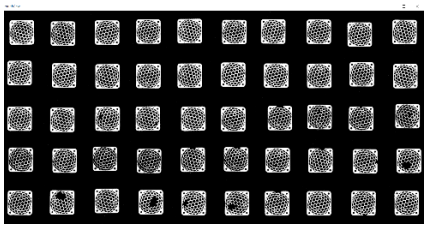

Figure 3: after process 3, 4

STEP2(1) – DATA MODELING 1 [1_pca_regression.py]

- Firstly, PCA is used to find meaningful scores from 96100 columns. To obtain 90% of the cumulative variability of eigenvalues, 30 principal components are achieved.

- Secondly, these 30 PCs as regressors. The regression model was built (for better classification logistic regression was used instead of linear regression because it is a bi-label classification). The outcome (accuracy) varies depending on the distribution of the train-test set, and the average was 80%.

- The same process for the greyscale images was done as well. The number of scores needed to create 90% of the cumulative variability was very similar as well as the exactly same evaluation result was achieved.

```
22 th:  0.8107932821847161
23 th:  0.8253903594055284
24 th:  0.8392155789644324
25 th:  0.8519833372514979
26 th:  0.8640827272221547
27 th:  0.8746453912880001
28 th:  0.8849339168478875
29 th:  0.8950686444934931
30 th:  0.9042124767645624
```

Figure 4: the cumulative variability of eigenvalues

```
[50 rows x 30 columns]
0 vs 0 -> O
1 vs 0 -> X
1 vs 1 -> O
1 vs 1 -> O
0 vs 0 -> O
0 vs 0 -> O
1 vs 0 -> X
0 vs 0 -> O
1 vs 1 -> O
1 vs 1 -> O
[[4 0]
 [2 4]]
ACCURACY: 0.8
```

```
[50 rows x 30 columns]
0 vs 0 -> O
1 vs 0 -> X
1 vs 1 -> O
1 vs 1 -> O
0 vs 0 -> O
0 vs 0 -> O
1 vs 0 -> X
0 vs 0 -> O
1 vs 1 -> O
1 vs 1 -> O
[[4 0]
 [2 4]]
ACCURACY: 0.8
```

Figure 5,6: evaluation result on the binary(left) and greyscale images(right)

STEP2(2) – DATA MODELING 2 [1_pca_controlchart.py]

- For building a control chart we decided to use Minitab. Since Minitab can have a maximum of 4000 columns, we resized the images to 50*50(=2500 columns). We decided to execute PCA (using correlation) only with no assignable defect data to find scores.

- Firstly, we used binary images as input of PCA. PCA is used to find meaningful scores from the 2500 columns of no defect data. Using 30 rows and 2500 columns, we executed PCA, found eigenvectors, and got 30 rows of scores. To obtain 80% of the cumulative variability of eigenvalues, 12 principal components should be achieved. To obtain 90% of the cumulative variability of eigenvalues, 16 principal components should be achieved.

	pc1	pc2	pc3	...	pc10	pc11	pc12
0	19.942624+0.000000j	6.709302+0.000000j	2.214386+0.000000j	...	-8.5325000+0.0000000j	12.974180+0.000000j	-6.5769940+0.0000000j
1	1.516344+0.000000j	-15.356674+0.000000j	-11.793168+0.000000j	...	11.313621+0.000000j	6.176332+0.000000j	-12.251894+0.000000j
2	25.141494+0.000000j	8.775755+0.000000j	12.424278+0.000000j	...	1.6090580+0.0000000j	-0.963464+0.000000j	3.5657410+0.0000000j
3	-12.658264+0.000000j	14.410222+0.000000j	-11.644549+0.000000j	...	-6.5497320+0.0000000j	-15.251424+0.000000j	0.0749660+0.0000000j
4	-21.929557+0.000000j	6.475222+0.000000j	10.541244+0.000000j	...	-27.853079+0.000000j	10.066788+0.000000j	-8.0530210+0.0000000j
5	-2.386796+0.000000j	11.186460+0.000000j	-10.989033+0.000000j	...	-9.2502500+0.0000000j	-1.787014+0.000000j	-1.3217780+0.0000000j
6	-31.063447+0.000000j	-2.094395+0.000000j	50.455186+0.000000j	...	6.7491310+0.0000000j	3.515670+0.000000j	1.9319080+0.0000000j

Figure 7: PCs for obtaining 80% of the cumulative variability of eigenvalues

- Secondly, to find scores of all 50 rows, we standardized the original 50 data with the mean and standard deviation of 30 no defect data. Multiplying this standardized data with eigenvectors of no defect data, we could obtain new scores of 50 rows.

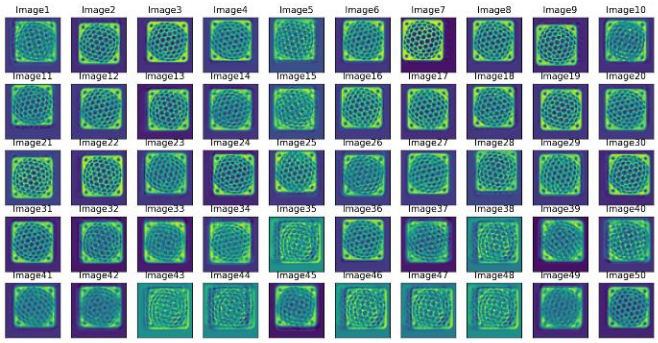

Figure 8: reconstructed images with 90% of the cumulative variability of eigenvalues

- Lastly, we reconstructed images with the new scores. Now we have assignable defect images that are reconstructed by no defect scores. The difference between the original images and these reconstructed images will be the main variable for the control chart.

- We have done the same process for the greyscale images as well. The number of scores needed to create both 80% and 90% of the cumulative variability were exactly the same as binary images. To obtain 80%, 12 PCs, and to obtain 90%, 16 PCs should be achieved.

```
8 th:  0.6631456164525914
9 th:  0.7045398267845537
10 th:  0.7422326210573287
11 th:  0.7729795036357191
12 th:  0.8031204476298418
13 th:  0.8316082630953527
14 th:  0.8553913286849005
15 th:  0.8757713799266554
16 th:  0.8955810507744311
17 th:  0.913119329047185
```

```
8 th:  0.6963549597692568
9 th:  0.7331237629307954
10 th:  0.7687821567549845
11 th:  0.7989806499929738
12 th:  0.8252177270478225
13 th:  0.8497122855343752
14 th:  0.8727319607270314
15 th:  0.8920295435999902
16 th:  0.9103153868662588
17 th:  0.9252134080050614
```

Figure 9, 10: the cumulative variability of eigenvalues with binary images as input(left) and greyscale images as input(right)

Cont' – DATA MODELING 2 [1_pca_controlchart.mpx]

** We used Minitab for this analysis. **

- In the beginning, we computed the mean of the intensity of all the pixels of a single image, in order to have one number summarizing all the information of a single image. It was necessary to find one single value for checking assumptions and building a control chart.

- Firstly, among binary input and greyscale input, we started with the binary images. The 16 PCs that explain 90% of the variability were used to build a control chart. More specifically, we used

the matrix of the difference between the original binary images and reconstructed binary images. After putting the data into Minitab, we first drew a scatterplot to see how these values trend.

- Then, we proceeded to check the assumptions for this single column of the mean. We used 24 out of 30 no assignable defect values for the training process because the other 6 are left for the test process. Normality test, runs test, and autocorrelation checks were done, and the assumptions were fine. Therefore, we could proceed to create the I-MR Control Chart.

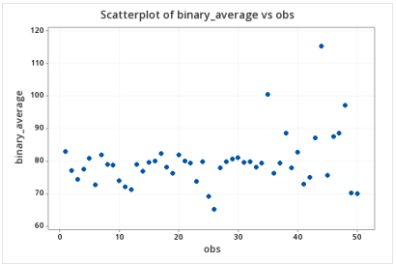

Figure 11: scatterplot of the mean of the binary data

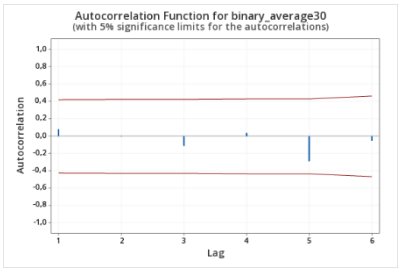

Figure 12: autocorrelation test on the mean of the binary data

Test

Null hypothesis H_0: The order of the data is random
Alternative hypothesis H_1: The order of the data is not random

Number of Runs

Observed	Expected	P-Value
13	12,67	0,886

The p-value may not be accurate for samples with fewer than 11 observations above K or fewer than 11 below.

Figure 13: runs test on the mean of the binary data

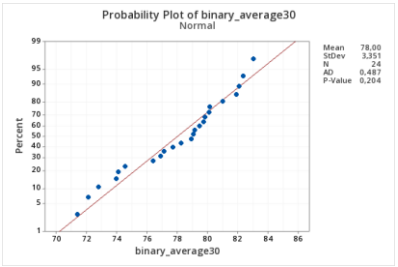

Figure 14: normality test on the mean of the binary data

- For creating the I-MR chart, we used 24 no assignable defect values for the training phase, and the rest 6 values and 20 assignable defect values were used for the testing phase. This model reached an accuracy of 65%.

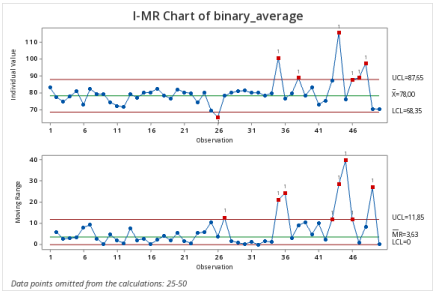

Figure 15: I-MR control chart on average intensity using binary data

- Then, to find the power of beta, we followed the same steps also for the binary data computed with the 12 PCs that explained 80% of the variability. This model reached an accuracy of 54%.

- Greyscale input, in this case, we couldn't use it because it was rejected by normality check. We tried to use box-cox transformation but it didn't work since we didn't have a minimum lambda to proceed.

Test

Null hypothesis H_0: The order of the data is random
Alternative hypothesis H_1: The order of the data is not random

Number of Runs

Observed	Expected	P-Value
15	13,00	0,404

Figure 16: runs test

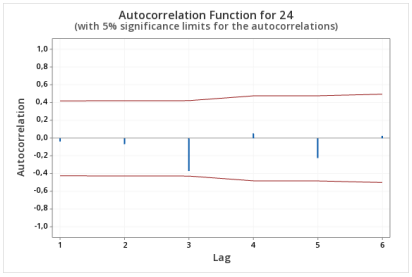

Figure 17: autocorrelation test

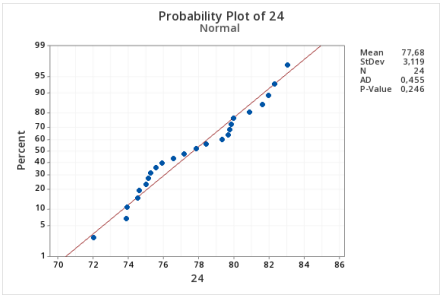

Figure 18: normality test

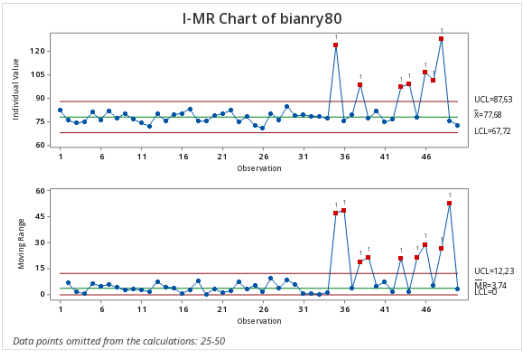

Figure 19: the I-MR control chart on the average intensity using binary data

STEP3(1) – DATA SPLITTING [2_makedata.py]

- Before using the ML/DL model, in order to avoid biased/overfitted outcome, the dataset ([30 data of label 0) & [20 data of label 1)) needs to be randomly and equally divided into 80% of the training set and 20% of the test set (For training set – [24 data of label 0] & [6 data of label 1], for test set – [16 data of label 0] & [4 data of label 1]). After this step, we will get a file named “5obj.npy”, which contains equally divided data.

- For both the ML/DL models, we tried K-fold cross validation (K from 1 to 5), but the validation process make accuracy even lower. We assumed that it is because the number of input data is critically insufficient. Therefore, in the end, we did not set up an extra validation process.

STEP3(2) – DATA MODELING 3 [2_modeling_randomforest.py]

- Among different machine learning models, we chose the **Random Forest model*.** The Random Forest model uses Decision Trees, which visually flow like trees. It starts with the root of the tree and follows binary splits based on variable outcomes until a leaf node is reached and the final binary result is given. This is why this model brings great accuracy for binary classification and why we chose this model.

- The accuracy of this model fluctuates drastically, but in most cases, it was below 60%. We can conclude that machine learning could not be effectively applied in this case, because the amount of data is extremely small.

STEP3(3) – DATA MODELING 4 [2_modeling_cnn.py]

** For this step, to achieve a better computational power of GPU, we used COLAB environment. To check this process, please open the colab.pdf file. **

- Among different deep learning models, we chose CNN (Convolutional Neural Network) model for the following reasons:

1. The type of input data is an image.
2. The location of the object in the input data changes over time.

- One of the early CNN models is **ALEXNET*** model, which brought huge progress in computer vision. For our model, we made several changes to the ALEXNET model. The main change was to exploit the binary classification method instead of the multi-classification method at the end of the model, so we applied binary cross entropy for the last layer. (Here we could also use categorical cross entropy if we tried to classify 4 types of defects.)

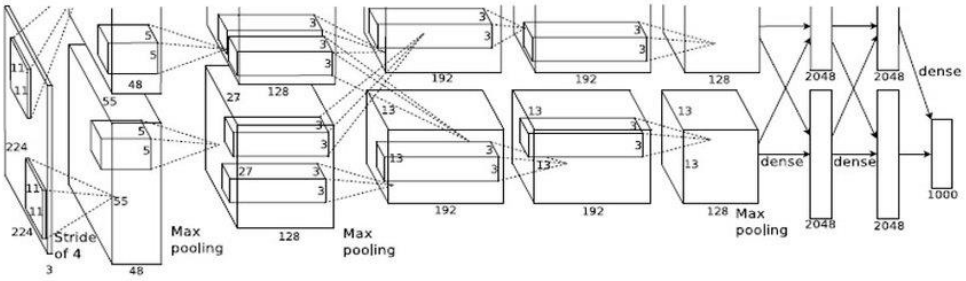

Figure 20: the structure of the ALEXNET model

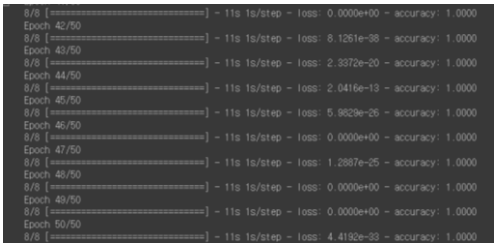

Figure 21: improvement of training accuracy

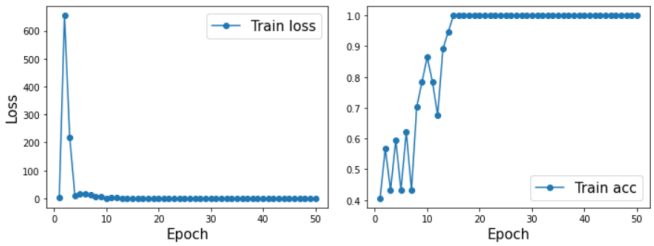

Figure 222: the graph of training loss and accuracy

- Training accuracy always reached 100% but test accuracy was an average of 70%. Again, due to the lack of data, it was really difficult to find a suitable parameter reaching 100% for the test accuracy.

Results and Thoughts

1. Between two PCA models

- For executing PCA, we found out that for input data either using the greyscale images or the binary images makes no meaningful difference, as long as we correctly binarized images saving features of a defect. However, PCA on the control chart model achieved less accuracy than on the regression model.

- There are several possible reasons why the control chart model had a low accuracy. At first, we had to resize the input images for the control chart model to use Minitab. Note that under the same condition – using image input of 50*50, PCA on regression model had similar low average accuracy with control chart model. Therefore, assuming we used the full images of 310*310 as input for PCA on the control chart, we would achieve better accuracy for the control chart model.

- However, we couldn't implement it with the full images because the python data type takes large memory. With 310*310=96100 columns of data, making 96100*96100 size of eigenvector matrix takes nearly 60GB of main memory. MATLAB could be better solution in terms of this problem. Hence, we realized the importance of input data and the software we use for analysis.

- Also, the mean might not be one best value to summarize 2500 columns. As matter of fact, if we have picture with half white and half black area and compute the mean, the point of that value is inside the control limit. Even some significant damage could be ignored when turning image data into its mean. We tried to find another test variable but we couldn't find some values working for this case. The importance of finding correct test variable is also what we've learned from this project.

2. Between ML/DL models

- CNN model has better accuracy than the Random Forest model. The main reason why the CNN model was better is that the position of the object in the input data varied over time (because we didn't manually crop each image). CNN model could precisely detect each object regardless of the position thanks to the sliding window technique.

- In that sense, we assumed that if we manually crop each object from the input image to fix it at the exactly same position while pre-processing step, the result of some models can be improved a little bit. (Again, we realized the importance of pre-processing.) However, we, instead of having that benefits by manual-cropping, decided to have input control through process automation since we wanted to observe as much different possibility from different input images as we can.

3. Between PCA models and ML/DL models

- Our assumption that the ML/DL models will have a better result than PCA models turned out to be partly wrong. We expected that the DL model will obtain 100% of accuracy since the images are simple. (ALEXNET can properly classify 1000 classes of high-resolution images with 97% of accuracy.) The main reason that the ML/DL models didn't work better, as we assumed, was that the number of data was too small to understand the correct features and train properly.

- In a real-world situation, with enough data, the ML/DL Models would work better than any other models, but we learned that with a limited amount of data PCA can have great predicting power.

References:

https://opencv.org/ (OpenCV)

https://scikit-learn.org/stable/modules/tree.html (Decision Tree)

https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (ALEXNET)