A Demo of Microservice for Customized Faulty Product Detection System in Smart Manufacturing

Nitesh Bharot, Mirco Soderi, John G. Breslin
Data Science Institute, University of Galway, Ireland {firstname.lastname}@universityofgalway.ie

***Abstract*—Product failure detection in smart manufacturing is important because it allows manufacturers to quickly identify and isolate faulty products before they reach the end of the production line. In today's fast-paced and highly competitive business environment, manufacturers need to quickly and accurately identify product failures to stay competitive and meet customer expectations. Previously the product quality was inspected manually and now it is examined using a machine learning algorithm to overcome the limitations of manual inspection. However, in the latter case AI and ML experts are required to do the task. Therefore, to overcome such dependency, this work proposes a microservice to allow the end user (an industry person, as well as an automated hardware/software agent) to test different combinations of data science and AI tools and technologies without any AI expertise. The proposed microservice exposes APIs that make it possible to select different combinations of feature selection, sampling, and classification algorithm. A demo environment with a Postman collection that includes few API calls to demonstrate how the proposed module enables customized selection to detect faulty products, is also discussed.**

***Index Terms*—Smart Manufacturing, Microservices, Industry 4.0, API, Product Fault Detection, Data Engineering**

I. Introduction

Low product failure rates are highly required to ensure a healthy production line within the industries. It is very common for products to be defective, which leaves things that are useless and cannot be sold or moved on to the next production stage. However, an increasingly competitive market, industries must strive for efficiency and effectiveness in their product development processes. Early detection of faulty products is a crucial factor in determining the overall success of a product. Early detection enables timely problem resolution, cost savings, improved quality, and faster time-to-market. However, late detection of faulty products will result in significant financial loss due to faulty quality products. Due to the frequent disposal of damaged products as waste, the high failure rate can also result in waste and needless energy use.

Therefore an efficient quality control plan is more and more important for production line management. Thus at the conclusion of the production process and occasionally throughout production, many methods are used to check the product quality [1]. The manual examination is frequently used for this, but it is less effective, more expensive, and time-consuming.

To address the issues associated with manual inspection, predictive analysis is becoming more and more prevalent across various application areas [2]–[4]. Additionally, identifying key variables in models can aid in pinpointing the underlying causes of defects [5], thereby enhancing the quality of future products.

A. Background

Building any fault detection model/prediction model involves four key steps: (i) Pre-processing: Raw data is cleaned by handling null and NaN values and scaled to a uniform range; (ii) Feature Selection: This data reduction technique improves dataset quality by selecting features that significantly impact prediction accuracy, reducing training time and resource utilization; (iii) Sampling: To address class imbalance issues in biased data, under-sampling or over-sampling techniques are employed, depending on the data type; (iv) Classification Algorithm: These algorithms assign predefined labels to input data for applications like anomaly detection, fault detection, and spam detection, and can classify data into multiple or binary classes.

Machine Learning (ML)/Deep Learning (DL) algorithms can potentially be able to early evaluate and predict product quality in a production line, as they can utilize the large amount of data generated by equipment in a production line and identify outliers that may indicate product failures [6]. This can be done without the need for extra modifications to the production line or additional labor. However, this also requires a dependency on the AI/ML expert by the industry.

In this demo we propose a microservice-enabled module that aims to provide a customizable solution to detect faulty products in the production line. The proposed system does not require any human element but can be operated by an automated hardware/software agent. This module removes the dependency on ML experts to accomplish the task. Simple API calls can be used to select the choice of feature selection method, sampling approach, and classification algorithm.

Further sections of this paper are structured as follows: Section 2 gives the overview of the proposed system, with its internal structure, and interface. Section 3 presents the step-by-step process to operate it. Conclusions are drawn in section 4.

II. Overview of proposed system

This section presents a customizable microservices-enabled early faulty product detection system in smart manufacturing. We characterize this system as customizable because user can customize their own mechanism to detect faulty products. The proposed system exposes APIs that make it possible to select different combinations of feature selection, sampling, and the classification algorithm. The proposed module improves the manufacturing process efficiency, accuracy, precision and recall without any ML expert intervention.

A. *Internal Structuring*

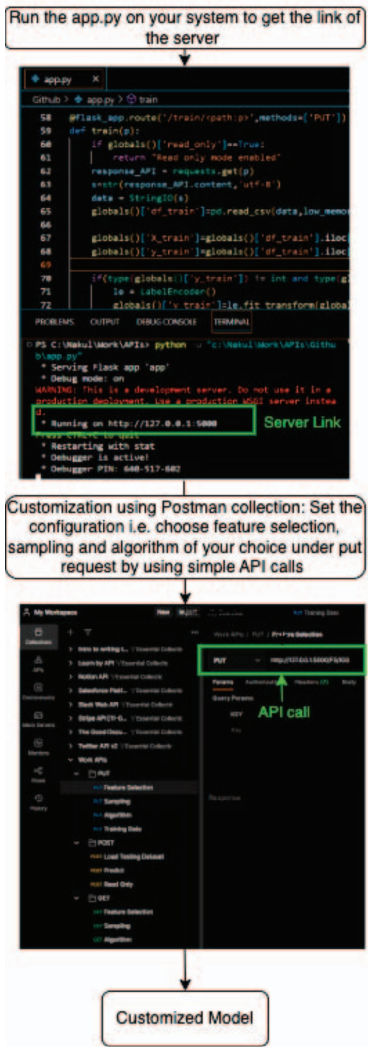
The proposed module is a microservice that can be easily containerized and used at any industrial site. Once deployed at the industrial site it enables the users to select feature selection, sampling and classification algorithms as per their choice with the help of simple API calls. Users can customize their combination of algorithms from the available techniques present in the microservice. This system allows individuals without a strong background in machine learning to easily customize and maintain it, without requiring extensive knowledge in the field. The software can be extended by adding further algorithms to be used at different stages of the data processing and classification purpose.

The proposed system is a microservice that can be accessed through simple API calls. It consists of three kinds of services: (i) set configuration, (ii) read configuration, and (iii) train and test. Further descriptions of these services are discussed in the following subsections.

1) Set configuration (PUT request): Under "Set Config", a selection is made by the user to customize his own method from the available list of feature selection, sampling and classification algorithms [7].

- Feature selection: User can set the feature selection method from the available list within the service. In the present state of the module, four efficient feature selection techniques are available such as; ANOVA (ANOVA), Information Gain (IG), Chi-square (CHI), Extra Tree Classifier (ETC), and No Feature Selection (NO) option. The feature selection can be configured through appropriate API calls where the algorithm to be used (ANOVA/IG/CHI/ETC/NO) and the required number of features are provided as input parameters.
- Sampling: To handle the class imbalance problem user can select the option for no sampling (NO), under sampling (UNDER), and oversampling (OVER). The sampling can be configured through appropriate API calls where, the technique used (NO/UNDER/OVER) will be provided as input.
- Algorithm: User can select the algorithm from the available list within the service. In the present state of module five classification algorithms are available to use and they are Support Vector Classifier (SVC), Multi-Layer Perceptron (MLP), Random Forest (RF), Gradient Boosted Tree (GDBT), RUSBoosted Tree (RUSBT) [7]. The algorithm selection can be configured through appropriate API calls where the algorithm to be used (SVC/MLP/RF/GDBT/RUSBT) should be mentioned as in the input.

2) Read Configuration (GET request): This service enables the user to identify the feature selection techniques, sampling methods and algorithms selected and applied to the dataset using appropriate API calls.


Run the app.py on your system to get the link of the server
app.py
Github > app.py > train
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Server Link
Customization using Postman collection: Set the configuration i.e. choose feature selection, sampling and algorithm of your choice under put request by using simple API calls
My Workspace
PUT
GET
POST
API call
Customized Model

```
@flask_app.route('/train/<path:p>',methods=['PUT'])
def train(p):
    if globals()['read_only']==True:
        return "Read only mode enabled"
    response_API = requests.get(p)
    s=str(response_API.content,'utf-8')
    data = StringIO(s)
    globals()['df_train']=pd.read_csv(data,low_memo

    globals()['X_train']=globals()['df_train'].iloc
    globals()['y_train']=globals()['df_train'].iloc

    if(type(globals()['y_train']) != int and type(g
        le = LabelEncoder()
        globals()['y_train']=le.fit_transform(globa
```

```
PS C:\Nakul\Work\APIs> python -u "c:\Nakul\Work\APIs\Githu
b\app.py"
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a
production deployment. Use a production WSGI server instea
d.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 640-517-602
```

Fig. 1. Demonstration of customization of proposed module

3) Train and Test: This microservice allows the user to upload the dataset for training and testing on the selected combination of techniques under "Set Config" and predict the output based on it.

4) Read-only (POST request): Security remains one of the major factors. Read Only API allows us to lock the selection and customization part for the selection of techniques. It only allows the user to predict values and disables the rest other API services.

III. Demonstration

This section presents the demonstration of the proposed module and shows how to use it. Figure 1 shows the major steps involved in the customization process of the proposed system and Figure 2 shows the training and prediction demonstration. Following are the major steps followed by the user to use the proposed module:

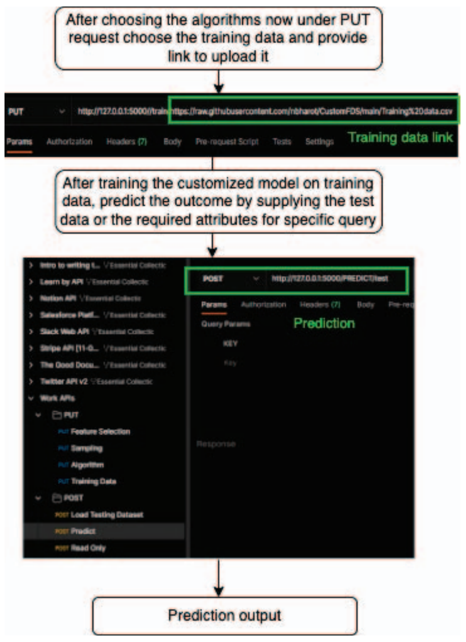

After choosing the algorithms now under PUT request choose the training data and provide link to upload it
PUT
http://127.0.0.1:5000//train https://raw.githubusercontent.com/nbharot/CustomFDS/main/Training%20data.csv
Params Authorization Headers (7) Body Pre-request Script Tests Settings
Training data link
After training the customized model on training data, predict the outcome by supplying the test data or the required attributes for specific query
Intro to writing t... Essential Collectic
Learn by API Essential Collectic
Notion API Essential Collectic
Salesforce Platf... Essential Collectic
Slack Web API Essential Collectic
Stripe API [11-0... Essential Collectic
The Good Docu... Essential Collectic
Twitter API v2 Essential Collectic
Work APIs
PUT
PUT Feature Selection
PUT Sampling
PUT Algorithm
PUT Training Data
POST
POST Load Testing Dataset
POST Predict
POST Read Only
POST http://127.0.0.1:5000/PREDICT/test
Params Authorization Headers (7) Body
Query Params
Prediction
KEY
Key
Response
Prediction output

Fig. 2. Demonstration of training and prediction in proposed module

- Download the proposed system microservice code and requirement.txt to be run from the GitHub repository[1].
- To run the API collection, you will need to import it into your Postman installation using the link provided in the GitHub repository. It has to be imported in a Postman installation to be run. To use Postman, it can either be downloaded for desktop or can be used directly over the web as well.
- Once API collection was imported in Postman, then under "Set Config" select the feature selection, sampling and classification algorithm as mentioned in section II-A1.
- After customizing the method, for training the model under "Train and Test" section upload the training data by providing the data link in the API call.
- Once the customized model is trained, then user can supply the test data to test the accuracy, precision, and recall of model.
- Once the model is customized and trained then to lock the customization, user can select the "Read-only" mode.
- Finally on the trained model by giving just the attribute values, user can predict whether it's a faulty product or not.
- Under "Read Config", we can check which feature selection, sampling, and classification is set by the user for training the model.

In addition to the research, some contributions for open source are made available in the GitHub repository for practical purposes such as requirement.txt file, Microsoft PowerPoint presentation, and postman collection consisting of few API calls.

[1]https://github.com/nbharot/CustomFDS

IV. Conclusion and Future Work

The proposed novel module enables user customization for choosing the mechanism for the early detection of faulty products in the manufacturing industry. The proposed system exposes APIs that make it possible to select different combinations of feature selection, sampling, and the classification algorithm. The proposed module allows the end user to test different combinations of these algorithms, and select the one that provides the best results in their specific case without the need of AI and ML expertise. Using simple API calls over POSTMAN user can select method of their choice within each service and customize their procedure for the detection of faulty products. Our research to have an impact, manufacturing industry can use the proposed module, and any automated agent in the system can use it, try different combinations, identify the optimal one, and use it for predictions. This overcomes the dependency on ML experts to build the ML models for the detection and classification of faulty products in the manufacturing domain. In future, a feature of transfer learning and other advanced AI/ML techniques such as LIME and Shapley for local model interpretability can be incorporated and used into the proposed module. So that user can use those services as well with the help of simple API calls.

Acknowledgment

This publication is supported by SFI under Grant Number SFI/16/RC/3918 (Confirm), and SFI 12/RC/2289-P2 (Insight). For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

[1] T. Brito, J. Queiroz, L. Piardi, L. A. Fernandes, J. Lima, and P. Leitão, "A machine learning approach for collaborative robot smart manufacturing inspection for quality control systems," *Procedia Manufacturing*, vol. 51, pp. 11–18, 2020.

[2] G. Köksal, I. Batmaz, and M. C. Testik, "A review of data mining applications for quality improvement in manufacturing industry," *Expert systems with Applications*, vol. 38, no. 10, pp. 13448–13467, 2011.

[3] A. K. Choudhary, J. A. Harding, and M. K. Tiwari, "Data mining in manufacturing: a review based on the kind of knowledge," *Journal of Intelligent Manufacturing*, vol. 20, no. 5, pp. 501–521, 2009.

[4] A. Kusiak, "Data mining: manufacturing and service applications," *International Journal of Production Research*, vol. 44, no. 18-19, pp. 4175–4191, 2006.

[5] S. Kang, E. Kim, J. Shim, W. Chang, and S. Cho, "Product failure prediction with missing data," *International Journal of Production Research*, vol. 56, no. 14, pp. 4849–4859, 2018.

[6] J. Leukel, J. González, and M. Riekert, "Adoption of machine learning technology for failure prediction in industrial maintenance: A systematic review," *Journal of Manufacturing Systems*, vol. 61, pp. 87–96, 2021.

[7] Z. Kang, C. Catal, and B. Tekinerdogan, "Product failure detection for production lines using a data-driven model," *Expert Systems with Applications*, vol. 202, p. 117398, 2022.