MÄLARDALEN UNIVERSITY
SWEDEN

PROJECT ASSIGNMENT 2.2

Module 2

THOMAS SELIN

School of Business, Society and Engineering
Course: Multivariate Data Analysis in Engineering
Course code: MTK337
Credits: 7.5 ECTS

Supervisor: Prof. Amare Desalegn Fentaye
Examiner: Prof. Amare Desalegn Fentaye

Date: 2024-10-21
Email:
thomasselin.studies@gmail.com

ABSTRACT

This project assignment is the third assignment in the course “Multivariate Data Analysis in Engineering”. The chosen problems to try to solve in the assignments of the course are related to difficulties in setting up efficient alerting for anomalies in IT (Information Technology) systems.

Solving, fully or partially, some or all of the defined problems would improve the operations of the system and can potentially also give insights to how the system itself can be improved.

The data that was used in this assignment consists of values extracted from metrics and logs from the system which is made up of many small components, called microservices. The dataset is limited to observations from one of those microservices.

The final model was successfully able to identify anomalies. Further ways to evaluate to which extent the problems are solved by the model are proposed.

Also, ideas of how to even better solve the problems are discussed and a plan on how to implement those ideas in later assignments is defined.

Keywords: autoencoder, anomaly detection, multivariate data analysis, machine learning, microservices, information technology

CONTENT

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

Abbreviation	Description
IT	Information technology
PCA	Principal Component Analysis - method used in multivariate analysis to model data and reduce dimensions
CPU	Central processing unit - computer hardware component that executes programs that for example processes data
RAM memory	Random Access Memory - computer hardware component that enables quick storing and reading of data

DEFINITIONS

Definition	Description
Microservice	A small component of an information technology system that handles one or a few tasks. Modern information technology systems tend to be partly or fully made up of microservices, and often a large number of microservices exist within such an IT system.

1 INTRODUCTION

This report is on the topic of using multivariate data analysis and machine learning to improve alerting when anomalies occur in an IT system. The report is structured as follows: introduction, materials, methods, results and discussion, and conclusion. The code that was used for model development can be found in Appendix 2.

1.1 BACKGROUND

The goal for the entirety of the course projects is to improve the alerting for anomalies in an IT system that is made up of many microservices. Microservices are most often small parts of a bigger IT system. A microservice often handles business logic, for example the calculation of the price of a product. It is common that microservices communicate and cooperate with one another to fulfill tasks.

There are many aspects of the alerting that would be valuable to improve, like better root cause analysis for incidents and more efficient anomaly detection, preferably as early as possible. This third project assignment of the course has centered around 4 main parts:

- Creating a suitable machine learning model.
- Evaluating the model and discussing to which extent it solves the problems (defined below).
- Discussion on how to further evaluate the model.
- Discussion on how to potentially create an even better model.

1.1.1 PROBLEMS

A few challenges with alerting in IT systems were identified as relevant to this project:

- Quickly becoming aware of serious anomalies.
- Quickly identifying where in the IT system a serious anomaly originates from.
- Quickly judge how severe the anomaly is.
- Minimizing the risk that more people than necessary get involved and spend time on troubleshooting an anomaly. This is more likely to happen if alerts for the IT system are imprecise so that alerts are set off in other parts of the system than where the problem originates from.

An important factor to be taken into account when creating a model is to which extent it is possible to explain the results that the model generates. Being able to understand the results well can give better insights regarding what causes anomalies in the microservice and by gaining those insights, appropriate actions can be taken to minimize those causes.

A model that doesn't use a lot of hardware resources during inference is highly desirable in this use case as the model could potentially be used for many microservices simultaneously.

1.2 Purpose and aim

With the logs and metrics from the system, be able to significantly improve the alerting for one microservice in the IT system. The same method could then be used to set up alerts for other microservices as well.

2 MATERIALS

In assignment 2, a dataset was extracted from logs and metrics of a microservice containing features and values of the type shown in Table 1 below. See *Appendix 1* for a further representation of the dataset. The data was selected by using domain knowledge of what are important features for how well a microservice is performing.

Discussion about what further features should be considered in later assignments that could lead to an even better model can be found in the *Results and discussion* section.

The data is somewhat sensitive and therefore not described in detail in this report. I consider the data trustworthy, and I didn't find any significant problems with the data throughout the entire process of assignment 2 or this, the third assignment.

The data was collected from the beginning of 2024-10-01 to, and including, 2024-10-07. That is a total of 7 days and was the maximum amount of data available at the time of collection. As it is most interesting to have good alerting and a performant model for the main usage hours of the system, which is 8.00-22.00 each day of the week, data from other times of the days were excluded. During 8.00-22.00 the usage is quite even. The total number of observations in the dataset after preprocessing was 5880. See *Table 1* below for a structured description of the dataset.

Variable name	**Variable description**	**Example value**
timestamp	Which minute the other values below belongs to	2024-10-03 13:00:00
warning_proportion	The proportion of all log events labeled with a log level that had the WARN log level	0.0022
error_proportion	The proportion of all log events labeled with a log level that had the ERROR log level	0.0013
avg_cpu_usage_percent	Average percentage of available CPU that was used	34
avg_memory_usage_percent	Average percentage of available RAM memory that was used	23

avg_disk_usage_percent	Average percentage of available harddrive disk space that was used	33
avg_latency_milliseconds	Average nr of milliseconds it took to process a request to the microservice	144

Table 1: Variables in the dataset

Labeled for the data, meaning data with values indicating whether or not the individual observations should have raised an alert was not available. This led to the choice of an unsupervised machine learning method. Target classes were whether or not an alert should be triggered.

3 METHODS

3.1 AUTOENCODERS

In the first two assignments no consideration was taken of the sequential nature of the data. Using a model type that can incorporate those relations between the observations in the data could potentially improve the model and was deemed as interesting to explore. One such type of model is autoencoders, especially certain variations of autoencoders.

[2]. The following section describes background related to autoencoders and considerations related to my use case. The entire section was sourced from www.claude.ai on 2024-10-20.

Autoencoders are neural network architectures designed to learn efficient data representations (encodings) by attempting to reconstruct their input data. They consist of an encoder, which compresses the input into a lower-dimensional latent space, and a decoder, which aims to reconstruct the original input from this compressed representation.

In the context of anomaly detection in sequential data, autoencoders can be particularly effective. By training on normal sequences, the autoencoder learns to reconstruct typical patterns efficiently. When presented with anomalous data, the reconstruction error tends to be higher, allowing for the identification of outliers or unusual patterns.

To capture temporal relations in sequential data, a specific type of autoencoder called a LSTM Autoencoder is often used. An LSTM model can maintain an internal state and learn dependencies across time steps, making them well-suited for sequential data analysis.

LSTM Autoencoders tend to be moderately quick at inference time while being excellent for anomaly detection, especially for complex patterns.

4 Results and Discussion

4.1 Summary of results from earlier assignments

There are in general low correlations between the variables, with the correlation between latency and memory usage as well as the correlation between error and warning proportions being the most significant. A PCA was conducted in assignment 2, which showed that the PCA was not particularly efficient at reducing dimensions in the data. It was therefore decided to use the original data set as input when creating the autoencoder instead of using the output data from the PCA.

4.2 Model creation and evaluation

After considering the superior anomaly detection efficiency and only slightly slower inference of LTSM models compared to the similar model type, Recurrent Autoencoders, I decided to proceed with developing an LTSM model.

To take advantage of the ability of LSTM autoencoders to capture sequential patterns in data I created a feature for the hour of the day for when each observation took place to provide temporal information to the model. The trained model can then be expected to relate the value of non temporal features to what is normal for that time of the day when determining whether or not the observation is an anomaly.

For further evaluation, it should be considered creating a few different versions of the models created in assignment 1,2 and 3. In the case of LSTM autoencoders, different sizes of the hidden state could be tested. Then evaluating these model variants on future real world data and gathering measurements for evaluation, including feedback from users of the model.

I used 70% of the data for model training. After the model was trained, I used 10% of the dataset for validation of the model. The results from the validation step were used when calculating a suitable threshold for the alerting. In the final step of evaluating the model I used the remaining 20% of the dataset for testing.

I experimented with different values for the number of epochs used in training and the size of the hidden state. The model seemed to perform well with a hidden state size of 6. Both the identified anomalies seemed to be correct and the total time for training, validation and test was low. Then by looking at the train and validation loss plot, I identified 140 epochs to be where the losses were sufficiently close to, but not yet, leveling out. See *Figure 1* for a plot describing the test and validation loss during training.

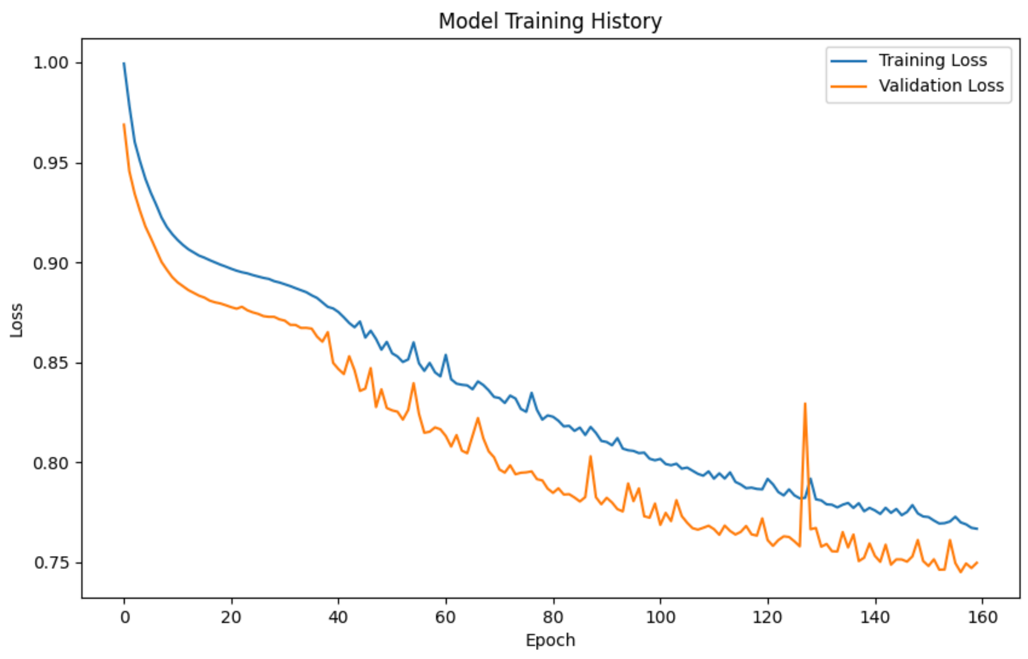


Figure 1: Plot describing the test and validation loss during training

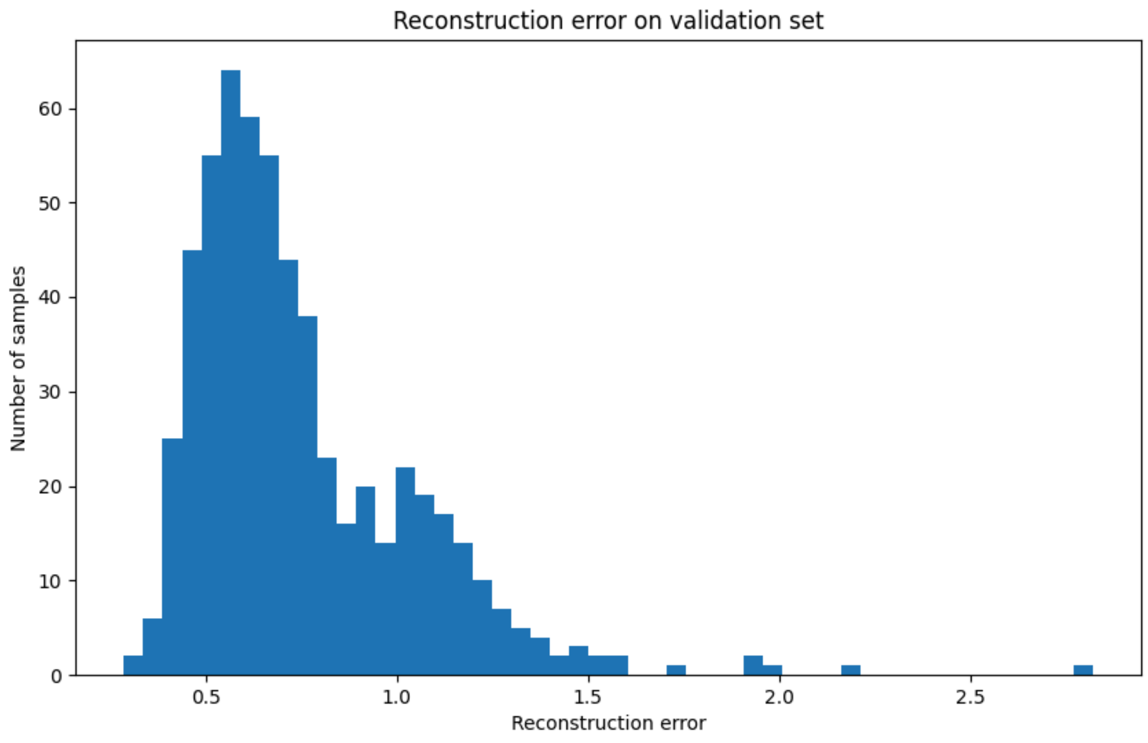
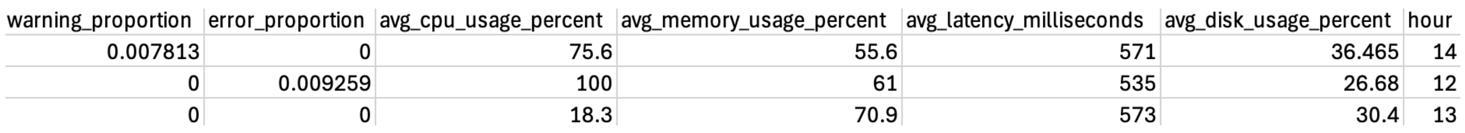


Figure 2: Plot of the reconstruction error on the validation data

I set the threshold to the 99.9th percentile of the validation set mean squared error reconstruction error as the data is not expected to contain a lot of anomalous data. See *Figure*

2 above regarding the reconstruction error. As an example, during one execution of the code, 16 anomalies were identified in the training data which contained a total of 1167 observations.



warning_proportion	error_proportion	avg_cpu_usage_percent	avg_memory_usage_percent	avg_latency_milliseconds	avg_disk_usage_percent	hour
0.007813	0	75.6	55.6	571	36.465	14
0	0.009259	100	61	535	26.68	12
0	0	18.3	70.9	573	30.4	13

Figure 3: *Example anomalies identified at inference of the model with the test data*

The first anomaly in *Figure 3* had significantly higher values than normal for warning proportion, CPU usage, latency and disk usage. The second anomaly had a significantly higher error proportion and CPU usage than what is normal. The third anomaly had significantly higher CPU usage, latency and also slightly high disk usage.

Considering the defined problems for the course project I found several promising aspect of the developed model, especially in comparison to traditional non-machine learning alerting approaches:

- **Quicker Identification of Serious Anomalies:** Detects anomalies across multiple variables simultaneously: Identifies patterns that might not trigger standard single-variable alerts.
- **Improved Pattern Recognition:** Recognizes anomalous patterns over time and in relation to the hour of the day.
- **Root Cause Analysis:** Can enable quicker identification of the anomaly's origin, particularly when the root cause is within this microservice.
- **Precision and Reduced False Positives:** Generates more precise alerts compared to non-machine learning based systems. Minimizes false alerts, reducing unnecessary troubleshooting time.
- **Explainability:** Provides good interpretability of model output. Allows examination of exact values for all features in an identified anomaly

By leveraging these capabilities, the model seems promising in improving anomaly detection efficiency and provides deeper insights into system behavior.

4.3 Plan for further evaluation

The main evaluation of the models must be done by the users, most often IT engineers, who after investigating the anomaly. Only they can give feedback if they conclude that a specific alert was valuable or not. The threshold for alerting could then be adjusted as wanted. In general, it can be said that if customers were significantly impacted by the anomaly then an alert would be considered valuable.

Just like in assignment 2, the identified anomalies contained some anomalies which indicated that the microservice was used very little during that specific minute. This could be because of uneven usage patterns from the user, network problems or problems in upstream microservices. For the latter two possible types of problems alerting might be done better elsewhere in the system and therefore it is likely that not all users of the model will want an alert to be triggered for this microservice, as an excessive amount of alerts can lead to desensitization to future alerts. Desensitization to alerts is a common and serious problem. Therefore it can be highly valuable to be able to ignore those kinds of anomalies. This is an interesting direction to explore further.

4.4 Potential improvement of the models

Including more features could potentially lead to a better model. Examples of features that could help are:

- Maximum CPU-usage
- Maximum memory usage
- Maximum disk usage
- Maximum latency
- Network related values, e.g. network throughput or packet loss [1]

Other possible ways to more efficiently solve the problems could be:

- Quantizing the model post-training to reduce computational requirements. [1]
- Implementing batch inference, as this can be faster than single-sample inference. [1]
- Adding features that represent the rate of change for key metrics, which can help identify sudden spikes or drops. [1]

5 Conclusions

The autoencoder models that were developed all proved capable of finding anomalies in the test portion of the dataset.

Plans for how to further evaluate, develop and use the model were described. For example, the model could potentially be improved by adding further features that represent the rate of change for key metrics.

The use of multivariate analysis and modeling using machine learning, in this case with an autoencoder algorithm, seems suitable for the use case of alerting for IT systems. The developed model seems promising for solving the defined problems, although a complete and accurate evaluation can not be made until the feedback from using the model on future data is collected.

The method and code used in this assignment can with no or little adjustment be used for alerting in other microservices as well. New data for a new microservice then needs to be collected and the model should be trained on that data.

One interesting way forward would be to build a transformer model. Transformers and autoencoders share significant similarities, such as encoding and decoding. Knowledge gained in this assignment could be further used in development of a transformer model.

References

1. www.claude.ai, 2024-10-21

2. www.claude.ai, 2024-10-20

APPENDIX 1: Representation of the dataset

This appendix gives a more complete representation of the dataset. As the actual used data contains sensitive information, the below data is made up data but has the same format as the actual used data and similar values.

timestamp	warning_proportion	error_proportion	avg_cpu_usage_percent	avg_memory_usage_percent	avg_disk_usage_percent	avg_latency_milliseconds
2024-10-03 13:00:00	0.0110	0.0060	32	32	44	129
2024-10-03 13:01:00	0.0010	0.0020	20	35	47	250
2024-10-03 13:02:00	0.0000	0.0001	4	11	35	70

APPENDIX 2: CODE USED FOR DATA ANALYSIS AND MODEL DEVELOPMENT

This appendix contains the code that was used to perform the data analysis. The code was written in the Python programming language.

```
import numpy as np
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, RepeatVector
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import matplotlib.pyplot as plt

# Data Import
dataset = pd.read_csv('final_dataset.csv')
print(f"Example observations:\n{dataset.head()}")

# Data Preprocessing
features = ['warning_proportion', 'error_proportion', 'avg_cpu_usage_percent',
'avg_memory_usage_percent', 'avg_latency_milliseconds', 'avg_disk_usage_percent',
'hour']
X = dataset[features]

# Split data: 70% train, 10% validation, 20% test
X_train_val, X_test = train_test_split(X, test_size=0.2, random_state=42)
X_train, X_val = train_test_split(X_train_val, test_size=0.125, random_state=42)

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Reshape data to 3D for LSTM [samples, timesteps, features]
timesteps = 10
def create_sequences(data, timesteps):
    return np.array([data[i:i+timesteps] for i in range(len(data)-timesteps+1)])

X_train_scaled = create_sequences(X_train_scaled, timesteps)
X_val_scaled = create_sequences(X_val_scaled, timesteps)
X_test_scaled = create_sequences(X_test_scaled, timesteps)

# Define the LSTM autoencoder architecture
input_dim = X_train_scaled.shape[2]
hidden_state_size = 6

input_layer = Input(shape=(timesteps, input_dim))
encoder = LSTM(hidden_state_size, activation='relu',
return_sequences=False)(input_layer)
repeat_vector = RepeatVector(timesteps)(encoder)
decoder = LSTM(input_dim, activation='linear', return_sequences=True)(repeat_vector)

autoencoder = Model(inputs=input_layer, outputs=decoder)
```

```
# Compile and train the model
autoencoder.compile(optimizer='adam', loss='mean_squared_error')
history = autoencoder.fit(X_train_scaled, X_train_scaled,
                    epochs=160,
                    batch_size=32,
                    shuffle=True,
                    validation_data=(X_val_scaled, X_val_scaled),
                    verbose=1)

# Function to detect anomalies
def detect_anomalies(model, data, threshold):
   reconstructions = model.predict(data)
   mse = np.mean(np.power(data - reconstructions, 2), axis=(1, 2))
   return mse, mse > threshold

# Calculate the threshold using the validation set
reconstructions_val = autoencoder.predict(X_val_scaled)
mse_val = np.mean(np.power(X_val_scaled - reconstructions_val, 2), axis=(1, 2))

# Visualize the reconstruction error
plt.figure(figsize=(10, 6))
plt.hist(mse_val, bins=50)
plt.xlabel("Reconstruction error")
plt.ylabel("Number of samples")
plt.title("Reconstruction error on validation set")
plt.show()

# Set the threshold to the 99.9th percentile of the validation set reconstruction error
threshold = np.percentile(mse_val, 99.9)

# Detect anomalies in the test set
mse_test, anomalies = detect_anomalies(autoencoder, X_test_scaled, threshold)

print(f"Number of observations in the test set: {len(X_test_scaled)}")
print(f"Number of anomalies detected: {np.sum(anomalies)}")
print(f"Percentage of anomalies: {np.mean(anomalies)*100:.2f}%")

# Print 5 of the anomalies with column names
anomalous_indices = np.where(anomalies)[0]
print("Indices of anomalies:", anomalous_indices[:5])
print("Anomalous data points:")
# Extract the original data for the anomalies
original_anomalous_data = X_test.iloc[anomalous_indices + timesteps - 1]
print(original_anomalous_data)
original_anomalous_data.to_csv('original_anomalous_data.csv', index=False)

# Visualize the reconstruction error on the test set
plt.figure(figsize=(10, 6))
plt.hist(mse_test, bins=50)
plt.xlabel("Reconstruction error")
plt.ylabel("Number of samples")
plt.title("Reconstruction error on test set")
plt.show()

# Plot training history
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Training History')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

MÄLARDALEN UNIVERSITY
SWEDEN