VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE
DEPARTMENT OF COMPUTATIONAL AND DATA MODELING

Bachelors Thesis

Implementation of application for visualization of regularities and randomness in data

Done by:
Audrius Baranauskas signature

Supervisor:
dr. Tadas Meškauskas

Vilnius
2021

Contents

Keywords **3**

Abstract **4**

Santrauka **5**

Introduction **6**

1 Signal processing and Recurrence plot **7**

1.1 Signal processing 7

1.2 Signal property categories 7

2 Web application development **8**

2.1 Analysis of analogous tools 8

2.2 Architecture 8

2.2.1 Project structure 8

2.2.2 Project workflow 9

2.3 Microservices 9

2.3.1 Front end microservice 10

2.3.2 Back end microservice 11

2.3.3 Plotter microservice 11

2.4 Recurrence plot module 11

3 Data classification model **11**

3.1 Training data 11

3.1.1 Choosing classification features 12

Conclusions and Recommendations **13**

Ateities tyrimų planas **14**

References **15**

Appendices **17**

A Pirmojo priedo pavadinimas **18**

B Antrojo priedo pavadinimas **19**

Keywords

Pateikiamas terminų sąrašas (jei reikia)

Abstract

Santraukos tekstas rašto darbo kalba...

Santrauka

Darbo pavadinimas kita kalba

This is a summary in English...

Introduction

Signals can be observer all around us. For example, measuring the time taken between a weight-driven pendulum clock's ticks produces a signal. It does not require a great deal of effor to image how such a signal behaves. We would expect the clock's pendulum to swing back and forth, each time travelling a minutely shorter distance until the pendulum stops completely. Analysis of even a part of such a signal can help us determine the pendulum's position far into future.

Now consider a more complex siganl: the rates of a stock market. People have been analyzing this data for decades, grasping to predict its future state. For the scope of this paper, we defined the term signal processing as *the science of analyzing time-varying processes* [14].

In this thesis we analyzed the non-triviality of digital sygnals. Certain signals can be classified as simple (relatively trivial), like the aforementioned clock's pendulum. A more complex (non-trivial) signal would be the rates of a stock exchange.

1 Signal processing and Recurrence plot

1.1 Signal processing

A signal is a function that conveys information about the behaviour of a system or attributes of some phenomenon [16]. For example, measuring the time taken between a weight-driven pendulum clock's ticks produces signal. In turn, for the scope of this paper, we defined the term signal processing as *the science of analyzing time-varying processes* [14]. By processing a signal we analyzed the non-triviality of a given signal. Analyzing a signal reveals that some signals have properties that can be categorized.

1.2 Signal property categories

We have considered the following categories:

1. Stationary and non stationary signals

2.

Signals have varying properties. Some consist of simple repetitions while others have no apparent patterns. For example, measuring the time taken between a weight-driven pendulum clock's ticks produces a relatively simple (trivial) signal.

2 Web application development

This project is aimed at creating a web application allowing one to interact with the recurrence plot algorithm in a user friendly manner. The project offers a feature of classifying data based on the generated plot using convolutional neural networks. This is an effort to further spread the popularity of this algorithm and help users intuitively grasp how it behaves.

2.1 Analysis of analogous tools

As of the date of publishing, only one tool was located capable of generating a recurrence plot online [15]. There are multiple implementations of the recurrence plot in Python as well as other languages, but none offer the ability to classify data based on the generated image.

It is noteworthy, that the aforementioned implementations require at least a minimal undertanding of software programming, a computing machine and specific software to compile and run the code. This is laborious and is not likely to attract new users to experiment with algorithm. Based on these factors, a decision was made to create a web based application that requires as little user knowledge to get started with the algorithm as possible.

2.2 Architecture

This project uses the microservices. Microservices are small autonomous services deployed independently, with a single and clearly defined purpose [12]. This design approach was chosen due to the flexibility and scalability associated with the architecture. The nature of microservices allows one to easily test, modify or out right replace each one of the components giving more freedom to the developer. The project ecosystem consists of the following microservices:

1. Front end web application
2. Back end for the web application
3. Python web server for plotting operations

Communication between microservices is performed via HTTP requests. In general, a query with JSON body is sent to a service and a JSON response along side an image attachment is returned. Figure 1 illustrates the microservice architecture of the project and data flow among services.

2.2.1 Project structure

The project is structured so that each microservice resides in an independent directory:

- app/
  - src/
    - components/
  - public/
- server/

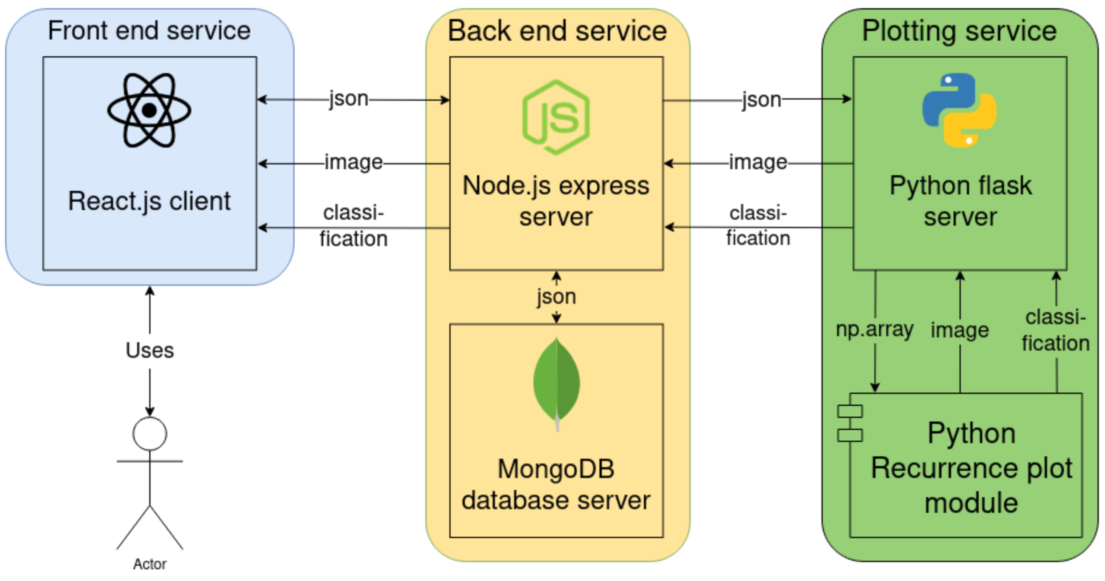

Figure 1. Microservice architecture structure

  - db/
  - public/
  - utils/
- plotter/
  - jupyter/

As the name suggests - `/app/` directory contains the front end ReactJS application. The NodeJS express[1] server resides inside the `/server/` directory. Meanwhile, `/plotter/` contains all of the Python source code. That includes the flask [2] web server, the recurrence plot module, jupyter notebooks for convolutional neural network model development and scripts for model training data generation.

2.2.2 Project workflow

We will now cover an example workflow of the application as per figure 1.

When first opening the app, a request is sent to the back end to fetch a list of existing plot data. The user selects an entry from the list and fills in remaining parameters for generating a recurrence plot. A request with select data ID is sent to the back end microservice. The back end service fetches data from the database and forwards it to the plotting service. The plotting service generates an image, then runs the image through a convolutional neural network to get classification data. Finally, the plotting service send the image along with classification data back to the back end service, which in turn forwards it to the front end service. The front end service displayes the image and classification data.

2.3 Microservices

The tools used for microservice development were largely open-sourced and relatively modern. Front end and Back end services were written in javascript based environments - React JS and

Node JS respectively. These choises were made due to the widespread use of javascript in modern web application development providing a large pool of open-sourced libraries and tools.

On the other hand python was the tool used to develop the plotting service. It is known to perform better on data handling and machine learning than javascript alternatives [13]. Both Python and Node JS have certain strengths and thus have appropriate community driven libraries and modules to reinforce their leverages in appropriate operations.

2.3.1 Front end microservice

The front end service is developed using React - A JavaScript library for building user interfaces [11]. SASS is used for styling the apllication due to the intuitive syntax it provides [9]. The microservice utilizes the Node Package Manager [6]. From the NPM registry, two open sourced libraries are used:

- node-fetch - A module that brings window.fetch to Node.js [5].
- query-string - a tool for building HTTP query string [8].

These libraries were used to facilitate communication via HTTP requests with the back end server.

Following the best practices of React development, the app is broken down into reuseable components. Figure 2 indicates the application structure denoting components with the standard JSX component notation `<component />`. We will be using this notation to refeter to JSX components.

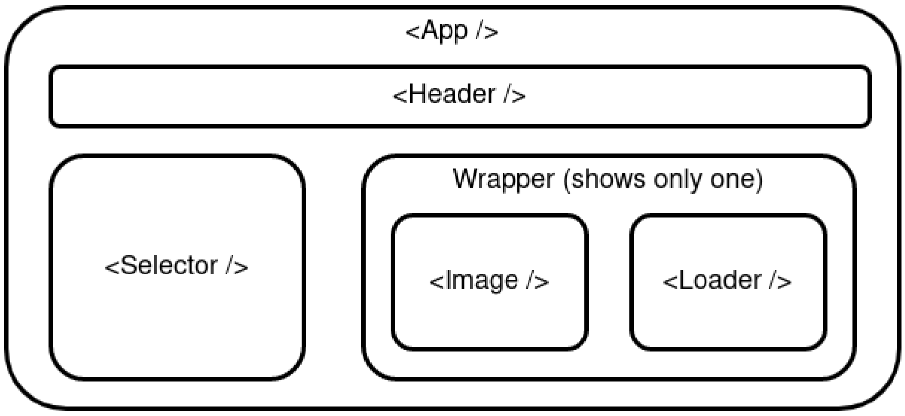

Figure 2. React app component structure

Figure 2 indicates that a root `<App />` component wraps the whole application. Initially, only the `<Header />`, `<Selector />` and `<Loader />` components are visible to the user. The `<Selector />` component sends an HTTP GET request to the backend service to retrieve a list of available plot data. This list is displayed inside the `<Selector />` for the user to pick from. A user must select a data entry and may add optional plotting parameters. Submitting the `<Selector />` form sends an HTTP GET request to the back end service. The backend service returns a JSON with the location of the generated recurrence plot image and additional parameters. After handling the server response - the `<Loader />` component is replaced by the `<Image />`. During any further plot requests, the `<Image />` is briefly replaced by the `<Loader />` component to indicate that a request is being processed.

2.3.2 Back end microservice

The backend microservice also utilizes libraries provided by the Node Package Manager. The service runs on an Node JS express server [1]. The server handles all requests from the front end service. Server endpoints cover the following operations:

- CRUD operations for plot data stored inside the MongoDB database
- Requests to generate a recurrence plot using the plotting service

The express server communicates with the database server by making use of an open sourced MongoDB object modeling library - mongoose [4]. The service itself does not generate any plot data, but merely acts as an intermediary between the front end service, the MongoDB database and the plotting service.

2.3.3 Plotter microservice

The plotter microservice handles requests to generate and classify recurrence plots. The service consists of 3 main parts:

1. Flask - a python web framework
2. Recurrence plot module
3. Data classification model

The flask service handles HTTP requests with JSON data as input. The service processes the input and generates an image using the recurrence plot module. Image is passed through the convolutional neural network to get the image classification. An HTTP response is then sent containing the classification data and the generated image as an attachment.

The innerworkings of recurrence plot module and a convolutional neural network are covered in detail in sections to come.

2.4 Recurrence plot module

The recurrence plot module is a Python implementation of the algorithm used to generate a recurrence plot. The module utilizes the numpy[7], scipy[10] and matplotlib[3] open sourced libraries.

3 Data classification model

A recurrence plot reveals certain information about the singal. After some practice a human can identify whether a given signal exhibits signs of periodity and / or stationarity, has a trend or seems to be random in nature. The goal of this model is to determine some the aforementioned characteristics of a signal by analyzing the reucrrence plot generated by it.

3.1 Training data

It is common knowledge that one requires data to train a convolutional neural network. The accuracy of a data model heavily weighs on the quality of training data and labeling. After a brief search for publicly available, labeled and categorized data suitable for training a recurrence plot model, decision was made for this data to be generated synthetically.

3.1.1 Choosing classification features

Before generating the data it is important to recognize what the convolutional neural network is expected to learn from it.

A choise was made to classify plot images into one of the three groups:

- Whether the data is stationary
- Whether the data has a trend
- Whether the data has is periodic

This decision was based purely on the ability to generate data that is labeled accordingly.

Conclusions and Recommendations

Išvados bei rekomendacijos.

Ateities tyrimų planas

Pristatomi ateities darbai ir/ar jų planas, gairės tolimesniems darbams....

References

[1] Expressjs - nodejs web application frameowrk.
https://github.com/expressjs/expressjs.com, 2021. [Online; accessed 2-Jan-2021].

[2] Flask - python web application framework.
https://flask.palletsprojects.com/en/1.1.x/, 2021. [Online; accessed 2-Jan-2021].

[3] Matplotlib - a comprehensive library for creating static, animated, and interactive visualizations in python.
https://github.com/matplotlib/matplotlib, 2021. [Online; accessed 3-Jan-2021].

[4] Mongoose - a mongodb object modeling tool designed to work in an asynchronous environment.
https://github.com/Automattic/mongoose, 2021. [Online; accessed 3-Jan-2021].

[5] Node fetch - a module that brings window.fetch to node.js.
https://github.com/node-fetch/node-fetch, 2021. [Online; accessed 2-Jan-2021].

[6] Npm - node package manager.
https://github.com/npm/, 2021. [Online; accessed 2-Jan-2021].

[7] Numpy - the fundamental package needed for scientific computing with python.
https://github.com/numpy/numpy, 2021. [Online; accessed 3-Jan-2021].

[8] Query-string - an http query string building module.
https://github.com/sindresorhus/query-string, 2021. [Online; accessed 3-Jan-2021].

[9] Sass - a preprocessor scripting language that is interpreted or compiled into cascading style sheets.
https://github.com/sass, 2021. [Online; accessed 3-Jan-2021].

[10] Scipy (pronounced "sigh pie") is open-source software for mathematics, science, and engineering.
https://github.com/scipy/scipy, 2021. [Online; accessed 3-Jan-2021].

[11] Facebook Inc. React is a javascript library for building user interfaces.
https://github.com/facebook/react, 2020. [Online; accessed 2-Jan-2021].

[12] L. Krause. *Microservices: Patterns and Applications: Designing Fine-Grained Services by Applying Patterns*. Lucas Krause, 2015.

[13] Miśtal Krzysztof. Performance comparison: Javascript vs. python for machine learning.
https://dlabs.ai/blog/performance-comparison-javascript-vs-python-for-machine-learning/, 2020. [Online; accessed 2-Jan-2021].

[14] R.G. Lyons. *Understanding Digital Signal Processing*. Prentice Hall professional technical reference. Prentice Hall/PTR, 2004.

[15] Norbert Marwan. Recurrence plots and cross recurrence plots.
http://recurrence-plot.tk/online/index.php. [Online; accessed 2-Jan-2021].

[16] R. Priemer. *Introductory Signal Processing*. Advanced Series In Electrical And Computer Engineering. World Scientific Publishing Company, 1990.

Appendices

Dokumentą sudaro du priedai: A priede ....

A Pirmojo priedo pavadinimas

Pirmojo priedo tekstas ...

B Antrojo priedo pavadinimas

Antrojo priedo tekstas ...