Predictive Model Creation Approach using Layered Subsystems Quantified Data Collection from LTE L2 Software System

University of Oulu
Faculty of Information Technology
and Electrical Engineering
Degree Program in Information
Processing Science
Master's Thesis
Jose Puerto
June 17, 2019

Abstract

The road-map to a continuous and efficient complex software system's improvement process has multiple stages and many interrelated on-going transformations, these being direct responses to its always evolving environment. The system's scalability on this on-going transformations depends, to a great extent, on the prediction of resources consumption, and systematic emergent properties, thus implying, as the systems grow bigger in size and complexity, its predictability decreases in accuracy. A predictive model is used to address the inherent complexity growth and be able to increase the predictability of a complex system's performance. The model creation processes are driven by the recollection of quantified data from different layers of the Long-term Evolution (LTE) Data-layer (L2) software system. The creation of such a model is possible due to the multiple system analysis tools Nokia has already implemented, allowing a multiple-layers data gathering flow. The process starts by first, stating the system layers differences, second, the use of a layered benchmark approach for the data collection at different levels, third, the design of a process flow organizing the data transformations from recollection, filtering, pre-processing and visualization, and forth, As a proof of concept, different Performance Measurements (PM) predictive models, trained by the collected pre-processed data, are compared. The thesis contains, in parallel to the model creation processes, the exploration, and comparison of various data visualization techniques that addresses the non-trivial graphical representation of the in-between subsystem's data relations. Finally, the current results of the model process creation process are presented and discussed. The models were able to explain 54% and 67% of the variance in the two test configurations used in the instantiation of the model creation process proposed in this thesis.

Keywords

Quantified Software Engineering, Machine learning, Layered Benchmarking, Data engineering, Data Science.

Supervisor

D.Sc., Professor, Mika Mäntylä

Foreword

Thanks for the almost staggering amount of support I received by everyone in Nokia. Thanks to my family and amazing friends for always believing in me, and helping me improve, I hope I can help you, support you, and give you at least half from all the positive energy I get from you.

Special thanks to my university supervisor Mika Mäntylä, who guided me into trying the correct approaches and listened to my cryptic explanations. To Kirsti Simula who actively corrected and guided this thesis work, and to Virpi Hanni for supporting me through all the thesis process.

We are more than what we offer, kinder as we are expected, and free to change what we desire, as long as we embrace the fear of the uncertain.

Oulu, Finland May 5, 2019

Jose Luis Puerto

Glossary

r^2 Coefficient of Determination. 65, 68, 70, 71, 75, 76, 79

3GPP 3rd Generation Partnership Project. 11, 24, 43

5G Fifth Generation Cellular Network Technology. 26

ASFFS Asaptative Sequential Forward Floating Selection. 30

BTS Base Transceiver Station. 17

CI Continuous Integration. 11, 12, 20, 39, 80

CPU Central Processing Unit. 9, 12, 22, 25

CV Cross Validation. 52–54, 71, 80

DL Down-link. 9–12, 15, 24, 37, 43, 65, 80

DSR Design Science Research. 14, 37, 38, 42, 43, 46, 48, 55, 77, 82

DSRP Design Science Research Process. 13, 14, 37, 47

E-UTRAN Evolved Universal Terrestrial Radio Access Network. 17, 18

EDA Exploratory Data Analysis. 55

eNodeB evolved NodeB. 17–19, 38

EPC Evolved Packet Core. 17–19

EPS Evolved Packet System. 18

EVS Explained Variance. 65, 70, 75

GA Genetic Algorithm. 30

hdf5 Hierarchical Data Format 5. 38

HPC Hardware Performance Counter. 10, 41, 44, 66, 72

HSS Home Subscription Server. 18, 19

HW Hardware. 11

IoT Internet of Things. 18

IP Internet Protocol. 19

IT Information Technology. 13

L1 Physical-layer. 17, 19

L2 Data-layer. 2, 9–12, 14–19, 22, 24, 25, 27, 33, 38, 48, 56, 77–80, 82

L3 Network-layer. 17, 19

LARS Least-angle Regression. 53

LASSO Least Absolute Shrinkage and Selection Operator. 31, 32, 47, 48, 52, 78, 79

LSTM Long-short Time Memory. 22

LTE Long-term Evolution. 2, 7, 9–12, 14–19, 24, 25, 33, 48, 77–80, 82

MAC Medium Access Control. 11, 20

MAE Mean Absolute Error. 65

ME Max Error. 65

MI Mutual Information. 29

MLP Multi-Layer Perceptron. 33

MME Mobility Management Entity. 18, 19

MSE Mean Squared Error. 65

OSI Open Systems Interconnection. 11, 17–19

P-GW Packet Data Network Gateway. 18, 19

PCRF Policy and Charging Function. 18, 19

PDCP Packet Data Convergence Protocol. 11, 19, 20

perf Linux Performance Analyzing Tool. 12, 22, 25, 38, 44

PHY Physical Layer. 20

PM Performance Measurements. 2, 9–12, 15, 16, 22, 24, 32, 33, 39, 40, 43, 55–57, 60–62, 65, 66, 71, 76, 79, 80, 82, 88, 89, 92, 93

QoS Quality of Service. 19

R&D Research and Development. 37

RFE Recursive Feature Elimination. 30

RFR Random Forest Regressors. 47

RLC Radio Link Control. 11, 20

ROHC Robust Header Compression. 19

RRC Radio Resource Control. 19

S-GW Serving Gateway. 18, 19

S1-c S1-control Plane. 18

S1-u S1-user Plane. 18

SAE-GW System Architecture Evolution Gateway. 19

SBS Sequential Backward Selection. 30

SCT System Component Testing. 12, 21, 38

SFFS Sequential Floating Forward Selection. 30

SFS Sequential Feature Selection. 30

SOI System of Interest. 9, 17, 19, 24, 80

SRI System Readiness Index. 21

strace Linux Syscall Tracer. 22

SUT System under Testing. 21, 38, 43

SVM Support Vector Machine. 28, 32

SVR Support Vector Regression. 51, 68, 70, 75, 78

SW Software. 11, 15

TLB Translation Lookaside Buffer. 25

tol Tolerance for Stopping Criterion. 52

UE User Equipment. 17–19

Contents

1 Introduction

The ability to understand the impact of new software functions and code modifications before its integration to a release and finished product is an always evolving theme appearing in various software engineering disciplines (Mens, 2008). This prediction provides a whole range of benefits such as a better planning of the future requirements, an accurate description of the quality attributes, and in general a valuable characterization of the system in development.

The discipline of software engineering focuses on the behavioral design of the System of Interest (SOI); this means software products most used quantified data reflects variables from the project rather than the SOI, these data being, for example, lines-of-code, problem reports, fault density, and others popular software's project related data (Muller, 2007, p. 1). Managing the design of the SOI by quantitative techniques, similar to other engineering disciplines, provides access to quantitative data that can measure the current state of the system accurately. This information enables precise and better estimation of future performances and hardware requirements, and it also enables comparisons of the different products or releases versions.

The rich quantitative data allows the creation of a prediction model, trained with the current and past state measurements of the system, with the particularity of being detailed quantified information incoming from different systems layers. These layers can be delimited based on a layered benchmark framework proposed by Muller (2007). This approach divided any software system into four layers, defined as hardware, operative system, services, and applications. It allows better analysis and understanding of the amount of affectation caused by each part to the complete system's performance, see Section 3.2.1. By having these data, the model can estimate and predict better relevant system performance aspects, based on the assumption, the more detailed information used in the analysis, the most likely one can reveal and understand more precisely complex and hidden relations in-between the system layers.

The use of the layered benchmark framework can enable a more in-depth comprehension of Down-link (DL) throughput performance, this one measured in symbols per second, that then depending on how many bits a symbol can carry, translates on bits per second. The DL throughput performance is one of Nokia's LTE system most critical defining characteristics that allow Nokia to differentiate from its competitors.

In order to apply this framework to this thesis work, and achieve the goal of revealing and increasing the understanding of the inner relations from the system's sub-layers and the DL throughput performance; the quantified data collected by the different test is organized in three kinds, first, thirteen DL throughput PM, second, function-level hardware performance data collected by profiling techniques (see Section 3.2.3), and last, events reception and processing times inside the L2 system by tracing techniques, see section 3.2.4. All these multiple properties are represented in quantified variables and are calculated on tests runs inside the L2 capacity testing environment.

Knowing that there has been successful work using machine learning in capacity testing environment to estimate Central Processing Unit (CPU) loads using test parameters(i.e., the different conditions and hardware configurations of the system, this one being used various final users) (Piippo, 2018, p. 87). Moreover, the work done in anomaly detection,

using the same kind of function-level hardware performance data that is used in this thesis, showing that high-dimensional function-level Hardware Performance Counter (HPC) (i.e., function-specific hardware event attributes separately) has better predictive power to characterize software performance change, when compared to lower dimensional data (Laivamaa, 2019, p. 75). Building upon these researches, this thesis attempts to characterize (predict) relevant DL throughput metrics, using machine learning on the L2 capacity testing environment, using high-dimensional data from function-level and adding another layer of high-dimensional data from events-level reception and processing times.

1.1 Problem Definition

There exist many variables affecting the final performance of the LTE L2 system developed by Nokia. It is known that the relatively little overhead of functions in the code can add up to generate noticeable degradation of the transmission speed rates. Being able to follow the releases trend and the impact to the PM related to the DL throughput using a multi-variable analysis, can be a great addition to be able to predict and estimate from the code fragments the expected behavior to fulfill the expected requirements related to the throughput, response time, and performance.

To draw a concrete picture of the practical use of this work in LTE L2 software design, one can think about a new chunk of code, this reflected in a new revision on a software version control system, that before being integrated to a critical branch or release, is quantitatively measured in regards of the effect of such code to PMs that measure the overall DL throughput performance. This precise measurement of the impacts by the new code allows a better estimation of the actual distance from the current system behavior to the optimal system performance and a better prediction of the future additional hardware resources needs.

1.2 Contribution

The objective of the thesis is to provide an experience of performance prediction and characterization using quantitative data gathered by diverse tools, coming from different dimensions of the LTE L2 system, an approach Nokia wants to explore, in addition to the many existing research done inside the L2 teams, with the goal of continuously leading up to new, better, and more precise analyze techniques.

This research is building on top of other thesis workers research as well as elaborating on them, attempting in doing the instrumentation and integration of all the parts needed to have a better understanding and accuracy in the prediction of the DL throughput affectation, as seen through software revision's evolution in time. The expected outcome is a unified process, beginning at the collection of different kinds of data and ending with improved accuracy in the prediction of the DL throughput related PM of the system.

This thesis deals with pre-processing, transformations, storage and availability of data in useful formats, visualization alternatives for high dimensional data, continuous integration process updating models as new samples are collected, as well as, fast and practical

use of the most adapted machine learning techniques to processes all data, having an outcome an accurate predictive model.

This type of studies can be categorized in the field of quantified software engineering, a field where there is not much research done as the primary focus. In other fields, there are various approaches to quantify the impact of software. However, studies quantifying the impact of the software exist. State of the art from different fields has been done, with particular emphasis in reliability and readiness of software.

In Nokia, there have been successful work done over the characterization of L2 hardware design using machine learning and artificial intelligence techniques in software and hardware capacity testing result data. The work from Piippo (2018, p. 87) showed excellent results in estimating CPU loads of hardware based on L2 parameters, creating a model that was able to explain 99% of the variation on CPU loads. The main contribution of this research is to perform similar characterization work in L2 software and thereby provide possible improvements for detailed design.

The quantified performance information from Software (SW) function level behavior can enable a better design of Hardware (HW) related requirements as the later can be modeled with a robust estimation allowing future needs to be taken into account quickly. Additionally, this unified quantitative information collected from different layers at multiple states of the project will enable both release-to-release and product-to-product comparisons.

1.3 Scope of the thesis

Nokia being a large organization provides LTE infrastructure to various services providers all around the world, they thrive to provide always the best quality service offering, to achieve this goal having an outstanding DL throughput performance with low latency is essential. A common ideal in any continuous software development process is that new feature updates should not cause degradation to the service quality, this ideal can be better achieved by specifying quantitative means to not only know if there is degradation but also how much there is. The thesis work has a big emphasis on understanding and predicting DL throughput performance. This under the scope of L2, from the Open Systems Interconnection (OSI) model, as specified by the 3rd Generation Partnership Project (3GPP), which consists of the Packet Data Convergence Protocol (PDCP), Radio Link Control (RLC) and Medium Access Control (MAC) sub-layers (3GPP 36.321, 2019; 3GPP 36.322, 2018; 3GPP 36.323, 2019).

The LTE L2 property of interest in this work is the throughput, i.e., the rate of successfully delivered messages through a given communication channel, quantified in bits per second. This rate is affected by the system components processing power, the behavior of the end-user, and especially crucial for this work, the different interactions between the L2 protocols.

It is pertinent to state that the related new features have an impact not only on the different PM but as well as, in the performance of the analysis tool itself. After obtaining a more accurate model about such impacts on different performance levels, this needs to be analyzed in the scope of the current Continuous Integration (CI) processes, with the

vision of integrating the findings of this thesis into the current software development pipeline.

The central thesis focus is the performance of the DL throughput whose performance is represented by its associated PM, this being one of Nokia's LTE system most critical defining characteristics that allow Nokia to differentiate from its competitors. It is essential to clarify, the data collected for the L2 software system comes from the development processes and inside the System Component Testing (SCT) environment. The data used to feed the proposed predictive model is limited by the tools that enable its quantified measurements, for example, the CPU performance at the functions level is limited to the number of registers in the hardware dedicated for this purpose. All data recollected by execution time tests are affected to a different extent by the recording of its results, and they are compared under the assumption that the overhead produced by the tests is the same for every data sample.

Test results and performance related data for different releases is available from the CI pipeline. The data comes from the SCT environment that has a different hardware configuration meant to fulfill the system requirements checking multiple scenarios of usage. The test cases from which the data will be collected have two levels, in one hand the function level, using Linux Performance Analyzing Tool (perf) (Benaissa, Bonvoisin, Feliachi, & Ordioni, 2016) to obtaining the information of how many cycles does a function use, and on the other hand the requirements level, obtaining information about the speed, the rate of transfer, and other alike, from different parameters and configurations of the system.

2 Research method

2.1 Design Science Research Process

This thesis consists of a state of the art review, and it finds its place in the six activities of Design Science Research Process (DSRP) as methodology proposed by Peffers et al. (2006). This methodology is best suited to research mainly done by the creation and evaluation of Information Technology (IT) artifacts (A. Hevner & Chatterjee, 2010), this being the case of this thesis, where the objective is to create an artifact that adds value, is useful, and can be evaluated. The DSRP has 6 activities represented in Figure 1 the activities consist of:

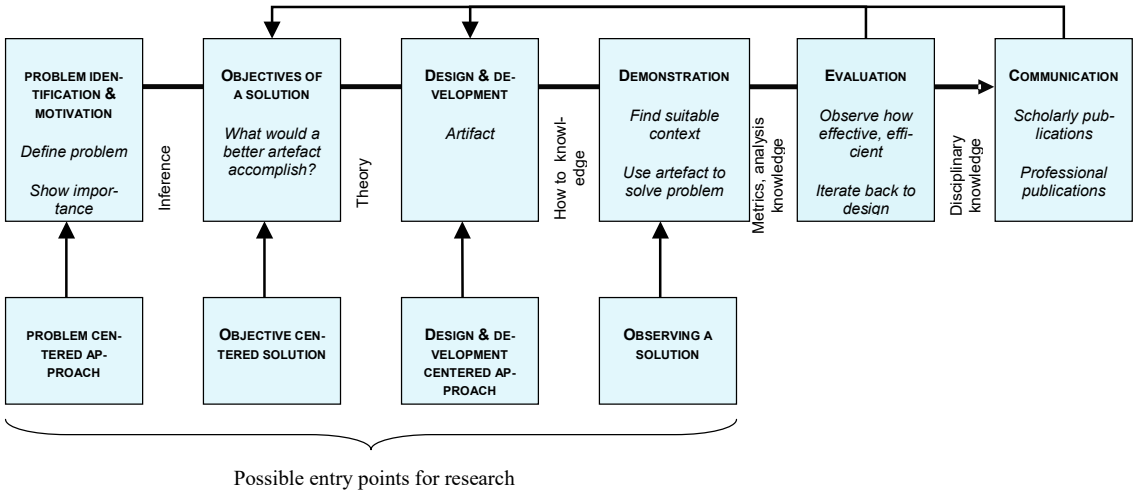

Figure 1: Design Science Research Process (DSRP) model, taken from (Peffers et al., 2006, p. 11).

- **Activity 1**. *The problem identification and motivation*: The problem definition is used to atomize the problem conceptually, so the solution reflects the complexity of the problem appropriately, this stated in section 1.1, also giving value to the contribution provides motivation and relevance to the research and the readers. This activity needs to be done knowing the current state of the problem and the relevance of its solution.

- **Activity 2**. *Solutions Objectives*: The objectives are inferred from the problem definition and shall be stated qualitatively or quantitatively, this way the produced artifact can be measured and if possible compared to existent solutions. For this activity is required to know the state of the currents solutions if they exist.

- **Activity 3**. *Design and development*: In this activity, the artifact's architecture and functionality are defined, and a first version of the artifact is developed. Knowing theories that can enable a solution is key to the transition from the defined objectives to the creating of the artifact.

- **Activity 4**. *Demonstration*: A proof of concept is done, for example, in a case study or a simulation, where one assesses that the artifact addresses the problem and going towards a solution. One needs to develop the knowledge to use the proposed artifact effectively

- **Activity 5**. *Evaluation*: The measurement that needs to reflect the quantitative or qualitative objectives defined in activity 2, this to be able to compare the evolution of the transformation of the artifact. Measurement is necessary in order to achieve the most efficient versions that would be rated better in regards to their ability to solve the main defined problem.

- **Activity 6**. *Communication*: Sharing the whole process of the artifact creation, allows to asses the rigor of the method, and allows other researches to build knowledge upon the one generated during the whole DSRP process.

2.2 Research Questions

The research question formulation is rather particular in Design Science Research (DSR), given its paradigm's distinct nature. Not all of the DSR publications have research questions. Recent research found out from 104 publications, 61.5% used at least one research question to link problem statements to their research approaches(Hoang Thuan et al., 2019, p. 20). In the same research, the authors proposed a typology of the articulation of different research questions mapped to the A. R. Hevner, March, Park, and Ram (2004, p. 7) Three view cycle. This is useful to clearly state research questions related to its location in between the interactions of DSR that enables knowledge to be built, this typology can be appreciated in Figure 2, having this in mind, the research questions for this thesis that intend to guide the design science research process for this thesis are the following:

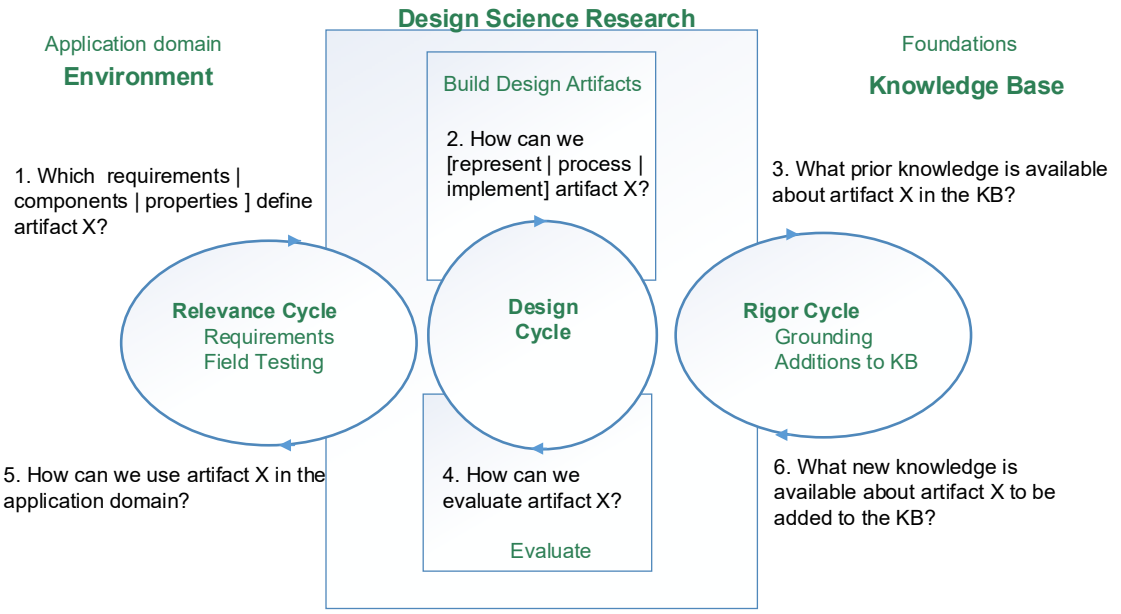

Figure 2: Design research genres with example research question structured inside A. R. Hevner et al. (2004, p. 7) proposed three cycle view, taken from (Hoang Thuan et al., 2019, p. 19).

- Main *RQ*: How can we implement a predictive model to fulfill the future needs based on the quantified data collected from different sources from the LTE L2 system?

- $RQ2$: What are most essential elements affecting the down-link throughput in LTE L2 system?
- $RQ3$: How can we evaluate and present the predicted model to ensure it fulfills the future needs?
- $RQ3.1$: How good can function-level performance data predict and model down-link throughput related PM?
- $RQ3.2$: How good can tracing signals performance data predict and model down-link throughput related PM?

These research questions are meant to guide the research of this thesis with the proposed artifact goals (Hoang Thuan et al., 2019). The research question one addresses the architecture of the process, research question two deals with the identification of most affecting element's relationships and finally, research question number three contains the usability goals of the artifact, in other words, to asses if the artifact does work and it is useful.

2.3 Cycles

The artifact proposed by this research is a system that models and analyses the possible impacts to the DL throughput with optimized SW and performance related parameters fine tuning in L2 SW. Following the iterative nature of design science and taking into account its seven guidelines to manage the creation process of the goal artifact (A. R. Hevner et al., 2004). The artifact produced in this thesis work is the process of gathering layered data into the creation of a predictive model, and its current state is achieved in 6 cycles:

Cycle 1. Define the union the different quantified data that will compose a complete sample

As the objective of this work is to produce an artifact that will be able to predict and reveal relations between different layers of the system, using as reference the layered benchmark approach proposed by Muller (2007) in the (still in writing process) paper on how to characterize software and hardware to improve the software design process. Three different types of quantified data from three layers were chosen to construct the sample; first, the DL throughput PM, second, function-level hardware performance data, and third, events reception and processing times.

Cycle 2. Define an architecture adapted to a continuous modeling and analysis process.

As is the importance of the demonstration of the real usability of any new created artifact is key to its real utility, the architecture of the processes, starting from the sample collections and its storage and, ending in a prediction output, shall be adapted to a developing environment, this meaning, that it is possible that tools, tests, the passing status from tests and functioning procedures are unavailable for short periods of time, something that is natural in a developing environment, this in-between a flux of modifications and improvements attempts will inevitably, sometimes, in its evolution process, generate

unexpected behavior that needs to be taken into account in the design of the artifact's architecture.

Cycle 3. Define transformations and pre-processing for all the particular raw data, that form a sample.

The raw data transformations are the key to enable proper performance of machine learning algorithms and techniques. Based on previous observations by other researches at Nokia, the first pre-processing options will be guided by this insight; then other pre-processing options will be considered and compared between themselves.

Cycle 4. Reveal and explore inside relations from the different layer's data by doing several exploratory analysis.

The use of exploratory analysis is vital due to the visualization of the data, displaying complex data high-dimensional data, in understandable human means, so the messiness or the smoothness of data can be recognized. An adapted graphical representation allows us to interpret data that in other cases, such as in one-dimensional graph (histograms or bar chart for example) will be perceived as noise. Noise can be smoothed by the use of multidimensional graphs as the scatterplot matrix, radar plot, stereo-ray glyph, and others (Yu, 2017).

Cycle 5. Create and compare different predictive models

The automated creation of models and its comparison based on the continuous collected and pre-processed data will provide trends information regarding the probable future affectation of the PM in the LTE L2 system.

Cycle 6. Evaluate the models and the architecture of the whole process, start making appropriate modifications.

In this cycle, the current state of the architecture of the whole process, the visualization decisions, and the pre-processing transformation will be continuously assessed and improved, modifying the previous cycles comparing the predictive power of the model in regards the decision taken on the previous cycles, in other words, the decision through the whole architecture is evaluated as a function of maximizing the predictive power of the generated models.

3 Background

The background section of this thesis is divided into five parts, first, an explanation of the LTE L2 software system, their specific properties and relations with the other OSI model layers, second, the research of the state of the art of similar quantified software engineering work, relevant insight coming from the software analytics domain and the exploration of the different software techniques and frameworks that combined make possible the model process creation and a possible better analysis and understanding of the systems behavior, third, the revision of the machine learning techniques and algorithms more adapted to the SOI from this thesis work and how they can be used to find relations in-between the system layers and predict them, fourth and final, the exploration of the most relevant data visualization methods to present the multidimensional data used at the whole process of this thesis.

These background section touches not only one domain but instead explores relevant papers and related domains intending to add insight or compare different approaches to the on-going design approach for this work and possibly support the quantified software engineering research domain.

3.1 LTE

LTE provides Internet access to mobile subscribers through the Evolved Universal Terrestrial Radio Access Network (E-UTRAN) and a Evolved Packet Core (EPC). the E-UTRAN is composed of one or several evolved NodeB (eNodeB) that have the role of Base Transceiver Station (BTS), these eNodeBs uses the relevant radio protocols to enable the connection of the User Equipment (UE) and the EPC (*LTE*, n.d.). These protocols performed by the eNodeB are found in the three first layers of the OSI model.

The LTE systems are composed of many subsystems, and naturally, each of these has their properties and relations in-between that affect the performance of the system altogether. This significant number of configuration possibilities in the structure of a LTE network instantiation creates the necessity of specific configurations simulation before actual the systems real construction and implementation.

It is relevant to explain the purpose of L2 on the complete communication system. The L2 layer following the OSI model architecture is in charge of providing functional and procedural means to establish, maintain, and release data links between the network entities, also in charge of permissions to transmit data between devices, identification and encapsulation of network layer protocols, as well as frame synchronization and error checking control (Zimmermann, 1980, p. 6), the L2 layer directly interactions with the Physical-layer (L1) and Network-layer (L3) as seen in Figure 3 (Dahlman, Parkvall, & Skold, 2011)

3.1.1 LTE Architecture

As stated in the previous section the LTE architecture consists of three main components; first, the UE that interacts with the E-UTRAN, second, the communication flow

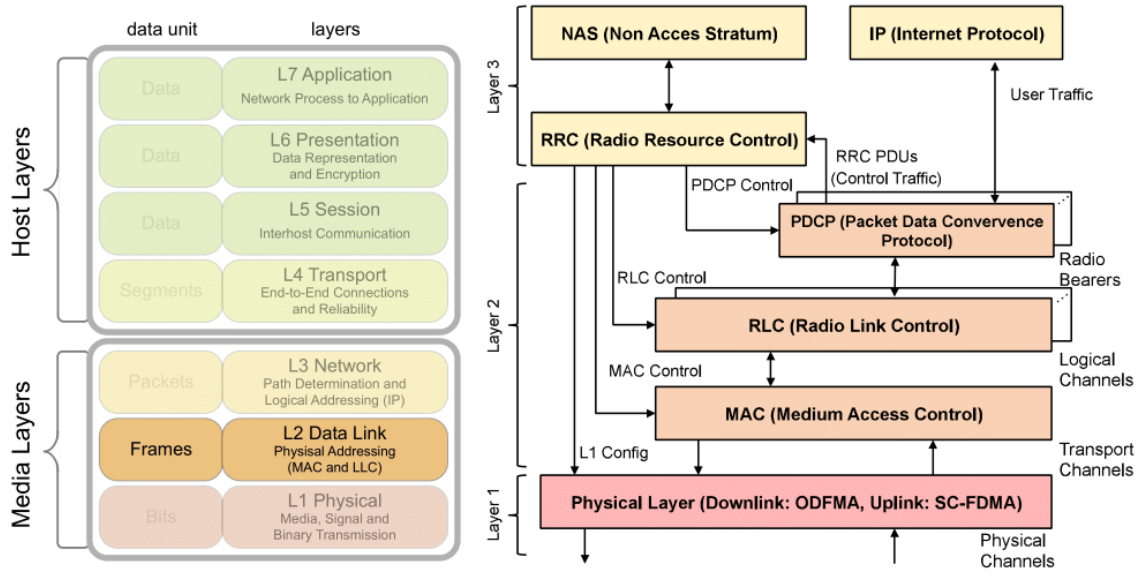

Figure 3: the OSI model view by layers (Offnfopt, 2015) and the L2 Data Link sub-layers.

continuous to the EPC, then it finally connects to the external networks that compose the internet (Dahlman et al., 2011), see Figure 4. These three components together make up the Evolved Packet System (EPS), which connected to any external networks, completes a LTE network.

The UE is a general term to refer to an end user device that connects to the network, such as smartphones, laptops, tablets, and all the increasing number of electronic devices connected to the internet following the trend of the Internet of Things (IoT). The user's UE connect to the LTE network by the E-UTRAN. The functions of the E-UTRAN are to manage all radio related processes and protocols, such as security, data delivery, header compression, retransmission, and mobility management. The E-UTRAN is made up of radio base stations; in this case, multiple eNodeBs, it is relevant to state that the E-UTRAN architecture is flat, meaning there are no central control nodes. As Figure 4 represents, the eNodeBs connection is dine by a X2 interface, enabling handovers through the different available nodes.

The eNodeBs are connected to the EPC by the S1 interface. This connection is split, in one hand by the S1-user Plane (S1-u), connecting to the Serving Gateway (S-GW), and in the other hand by the S1-control Plane (S1-c) that connects to the Mobility Management Entity (MME). It is relevant to point out that the eNodeBs can be connected to multiple S-GW and MME, as shown in Figure 4, but each UE connection exist only through a single connection of MME and S-GW (Dahlman et al., 2011, p. 110).

The principal control component of the EPC is the MME; its functions are to provide authentication and security to the UEs, as well as, keeping track of UE locations, by this handling subscription profiles, service connectivity, and mobility and bearer management. the component Home Subscription Server (HSS) is a database server, that contains subscriptions and location data from the registers users, by providing this information it supports the MME functions (Dahlman et al., 2011, p. 110) and (Piippo, 2018, p. 36).

The S-GW collects statistics and data for charging, and it acts, in one side, as the user plane connector for the eNodeBs with both the MME and the Policy and Charging Function (PCRF), and in the other side, it connects as control plane to the Packet Data Net-

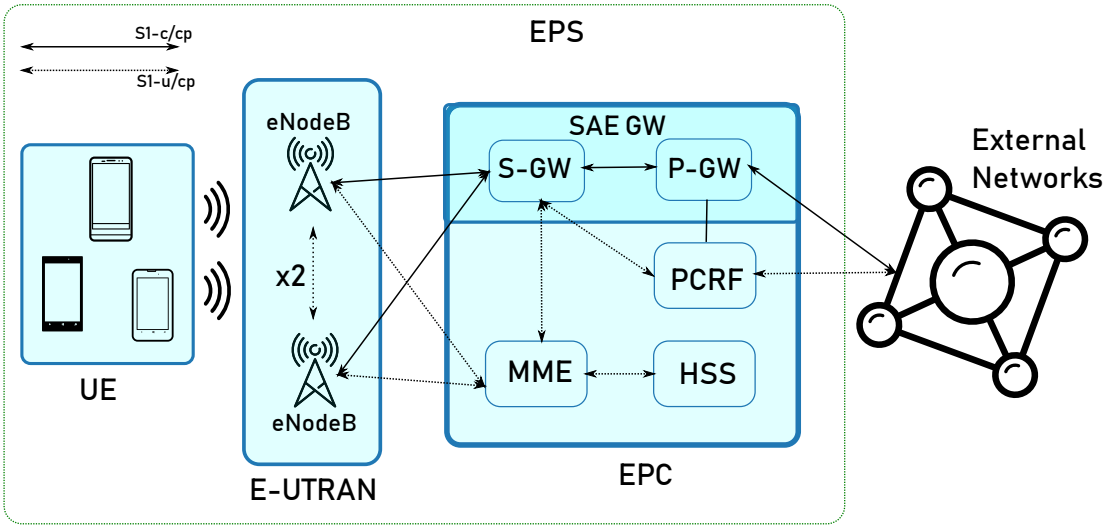

Figure 4: High level architecture of a LTE network

work Gateway (P-GW). This relays the data from the eNodeB and the P-GW, being an anchor whenever the UE changes its single eNodeB connection. The P-GW provides the Internet Protocol (IP) for the UE that will identify it on the external networks; it is in charge of information gathering for charging, traffic gating, and the enforcement of the Quality of Service (QoS).

After the brief explanation of two of the three components of the EPC, the MME and the HSS, the remaining component, the PCRF as its name suggests, is responsible for charging and also for the QoS. It is connected to both components of the System Architecture Evolution Gateway (SAE-GW), the P-GW and the S-GW, keeping them informed of QoS requirements for services and policies (Dahlman et al., 2011, p. 110)

3.1.2 LTE Radio Protocols

The L2 software is tailor-made as part of the LTE solution to Nokia's clients. It is relevant to explain the SOI in which the study is conducted and the predictive model addresses.

The L1 and L2 handles the LTE radio access protocol for the control plane and user plane. The L3 also handles the Radio Resource Control (RRC). The Physical layer, Data link layer, and the Network layer from the OSI model, as seen in Figure 3 have the scope of the radio protocols in LTE. The protocols manage the signal flow enabling the proper and correct connection between the UE and the EPC. Signaling for both control and user planes means managing packet data, The user plane handles the end user IP data traffic and the control plane handles mobility and other signaling related to connections creation and maintenance (*LTE*, n.d.; *LTE-Advanced*, n.d.; Holma & Toskala, 2009, p. 36–39)

To understand better the interactions of the protocols, an overall architecture can be seen in Figure 3 and Figure 5.

- The PDCP does IP header compressor using the standardized for mobile communications technologies header-compressor Robust Header Compression (ROHC). It is in charge of ciphering and, for the control plane, integrity protection of the

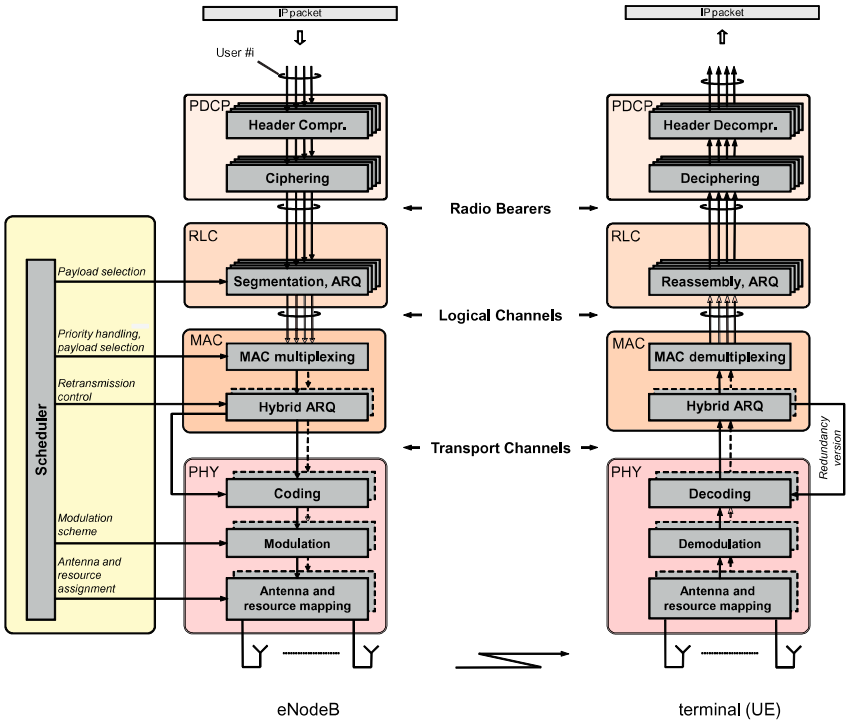

Figure 5: LTE protocol architecture taken from (Dahlman et al., 2011, p. 112).

transmitted data, as well as in-sequence delivery and duplicate removal for handover. At the receiver side, the PDCP protocol performs the corresponding deciphering and decompression operations. There is one PDCP entity per radio bearer configured for a terminal.

- RLC is responsible for segmentation/concatenation, retransmission handling, duplicate detection, and in-sequence delivery to higher layers. The RLC provides services to the PDCP in the form of radio bearers. There is one RLC entity per radio bearer configured for a terminal.

- MAC handles multiplexing of logical channels, hybrid-ARQ retransmissions, and uplink and down-link scheduling. The scheduling functionality is located in the eNodeB for both uplink and down-link. The hybrid-ARQ protocol part is present in both the transmitting and receiving ends of the MAC protocol. The MAC provides services to the RLC in the form of logical channels.

- Physical Layer (PHY) handles coding/decoding, modulation/demodulation, multi-antenna mapping, and other typical physical-layer functions. The physical layer offers services to the MAC layer in the form of transport channels.

3.1.3 The Testing Environment

As Nokia follows agile methodologies, the development is done in multiple small iterations; the objective is to have as well structured and easy to understand code as possible. One tool used is the Open-source automation server Jenkins (Moutsatsos et al., 2017) enables the practice of continuous integration CI thus the process of testing can be integrated fast and efficiently into the software production pipeline. An essential goal of the testing systems main goal is to test that the software and hardware are proper for

production and release. As for this research, the data is obtained performing capacity testing a part of the SCT method. The SCT method aims to verify components correct functionality separately, this by creating stubs or other types of simulating behavior software objects when external interacting components are not available.

The data is obtained from the System under Testing (SUT) at the layer two level as shown in Figure 6

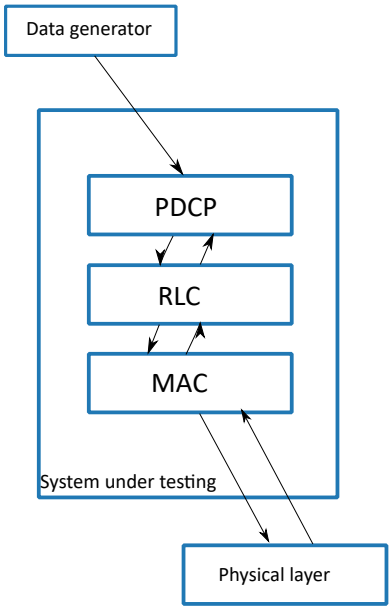

Figure 6: SCT testing environment

3.2 Quantified Data Modeling

What is quantified software engineering? As for the words tell, it is related to quantitative measurement of the application of engineering to the complete systematic cycle of software development. The process of representation from concepts in the theoretical world to quantifiable variables in the empirical, allow us to test propositions by hypothesis and eventually obtain well-grounded empirical knowledge about the studied phenomenon.

There is an always present gap filled with uncertainty in-between the plan and the actual execution of any software project (Fitzgerald & Stol, 2017). There are many techniques and approaches to assure performance; for example, one attribute often studied is how to measure test code quality and its impact to handle issues (Athanasiou, Nugroho, Visser, & Zaidman, 2014). There are also techniques to use Natural Language Processing to identify technical debt (Maldonado, Shihab, & Tsantalis, 2017).

There exists not much research with a primary focus on the quantification of software reliability and readiness (Asthana & Olivieri, 2009) where they state of quantitative criteria for software readiness and the need for a systematic approach to quantifying software goodness. The need of a System Readiness Index (SRI) and a method that presents different criteria coming from quality and reliability data in simple metrics and visualizations (Olivieri, 2012), has also contributed to the definition of a systematic approach to quantify Hardware and software(as a coupled system) goodness.

There is fairly recent work in the task of modeling a computer system to predict future resource usage as well to detected anomalies, collecting present state data such as memory usage and CPU statistics, having a base assumption that the past usage of resources influences the present and future resource usage (Schmidt, Niepert, & Huici, 2018). This is a similar assumption to the one chosen for this thesis work. In their paper, the researchers use the perf tool to sample system calls (equally as in this thesis work) and the Linux Syscall Tracer (strace) tool, tracing tools available on Unix systems. After the data is pre-processed, they use a Long-short Time Memory (LSTM) recurrent neural network, creating a model that maintains a hidden current state representation of the system. The end-to-end architecture of their process can be seen in Figure 7

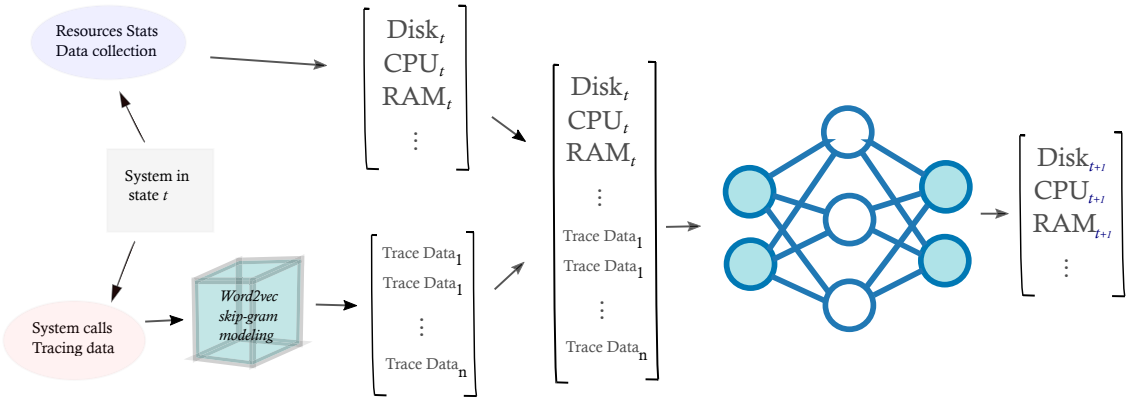

Figure 7: Example of resource usage prediction model architecture.

A particularity in their model design is the use of a representation learning approach for their tracing data. They manage it as a vector of event representations, this using *word2vec* skip-gram model (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013). In this thesis, the tracing tool is not strace but rather a tailor-made tracing tool used in Nokia. This tool provides already the average time of each event called during a specified period of time, also, because there is already insights observed by previous researches done here at Nokia, the model proposed in this thesis work will aim to feed the model with those previously identified events that affect the downlink throughput performance the most.

The domain of load testing in large scale systems is related to quantified software engineering by the need to organize large amounts of data and the final goal that is an indicator of the current state of the system, it maybe pass or fail, or more directly the current discrete value that allegedly reflects a vital aspect of the system(this is analogous to classification and regression differences). It is possible to automatically detect performance regressions, base on the analysis of performance counters organized in hierarchical clustering, this under the premise that during the test execution there would be specific conditions that will generate clusters on the performance counters vectors (Chen et al., 2017, p. 5–6). This in contrast to the approach of this thesis, where the granularity of the samples is the joint mean values from the profiling, tracing and PM variables, in other words, the referent axis for samples comparison in this work is revisions (who reflect the modification from new code fragments to the L2 system), rather than seconds during the execution time of each test (Javidian, Jamshidi, & Valtorta, 2019).

3.2.1 Layered Benchmark Approach

Measuring and analyzing the system at several levels improves the understanding of the system performance throughput. It is known that the entire system is too complex to understand in one single pass, therefore checking the layers or subsystems in a generic four-layer model enables the analysis of the complexity of the internal subsystems relationships (Muller, 2007), a representation of this model can be seen in Figure 8. There are four horizontal layers, plus one lateral and parallel to the others.

- The first layer in this framework, called hardware layer, comprises all the quantified data that is possibly collected from CPU performance, memory access, bus, cache access and misses, disk read and write, network times and others alike.
- In the operating system, layer data can be collected regarding the handling of interrupts, task switching times, resource management, process communication, and other OS services.
- The services layer in this framework[1] comprehends on services build on the performance of the other two layers below. These services can be, for example, networks access times, storage devices, database functionality, and others. This layer depends on quantified data of throughput, execution time, CPU speeds, memory footprint, cache usage and impact, number of interrupts and context switches, and amount of invoked OS services.
- At the applications layer, the performance of end-to-end functions is added; this is, as perceived by the user of the system. Once again, it depends on the layers below plus the affectation of the end-to-end functions that, is what the final user interacts with directly.
- Finally, the tools perpendicular layer This layer contains the affectation of linkers, compilers, configurators, high-level generators and alike. The previously mentioned tools influence most other layers, typically. Quantified data that can be collected at this level is locality and density of code, the efficiency of the generated output and run-time overhead introduced by the tools (Muller, 2007).

[1] this is the name given by Muller (2007) refers to the third layer in the proposed layered framework

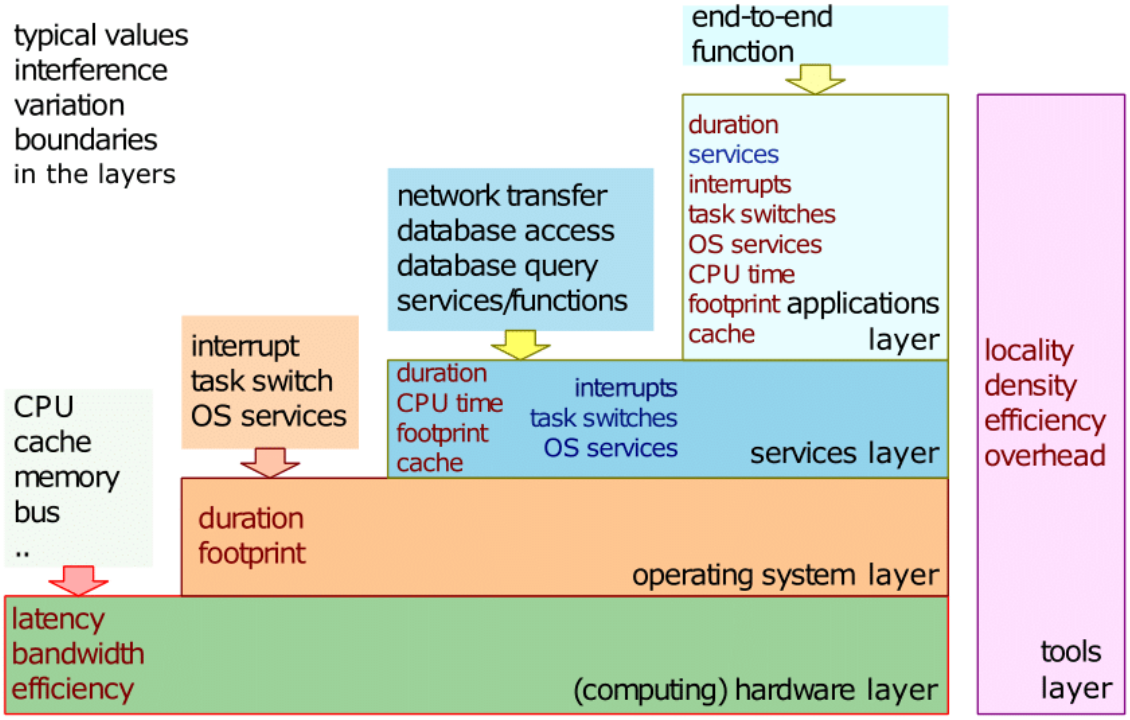

Figure 8: Graphic representation of the layered benchmarking approach taken from (Muller, 2007, p. 11).

3.2.2 Performance Measurements

In the case of the SOI to this thesis, the Nokia's LTE L2 system, the quantified data coming from the services layer (following the layered benchmark approach) are custom performance measurements (PM). These measurements record fundamental properties emerging in different contexts. It is essential to explain that there is a great amount of PM required to be implemented by the 3GPP (3GPP 36.425, 2019), and in this thesis, only 13 PM are taken into account, these ones related to the DL throughput.

There exists similar research on how to determine the most affecting performance counters [2] which can be done by first removing performance counters that seem to be measuring the same property of the system, this can be quickly done by checking for zero variance in different versions, second by applying clustering techniques, in this manner grouping the similar ones and mapping each of the resulting clusters to a system's performance model (Shang, Hassan, Nasser, & Flora, 2015, p. 18). This is going to be very relevant to this thesis, as, with the results of the first cycle, there is a need to review the real differences of the 13 PM, see Section 5.1.

3.2.3 Profiling Technique

Software profiling is a technique used to dynamically study the behavior of a program in terms of frequency of the function calls or the cost of instructions with regards to proces-

[2] performance counters is the most common name used in the literature, in this work, the analogous data is called performance measurements (PM)

sor time consumed or other hardware metrics, such as L1 cache Translation Lookaside Buffer (TLB) misses on the CPU. Different types of profilers exist, depending on the method used to gather data. For example, instrumentation profilers depend on special instructions inserted by the programmer or the compiler to collect data, or by running the code under the control of the profiler. Instrumentation profilers, who need to add special instructions to the instruction stream, usually can affect the performance of the system being measured, which needs to be taken into account when using such tools.

Another standard methodology of collecting data for profiling is by sampling in intervals determined by hardware events, like a real-time clock or performance counters, in other words, the clock or the counter is configured to cause an interrupt when an overflow occurs, at which point the profiler will ”take a sample” by recording the state of the program when the overflow occurred (e.g., the instruction last executed). With a high enough amount of samples, we can get an excellent approximation of which parts of a program are more expensive with regards to time spent (if a clock is used), or hardware events measurable by a performance counter, which can be cache TLB or cache misses, flushes, instructions executed, missed branches on the branch predictor, et cetera. The underlying processor of the system determines the performance counters available.

Sample-based profilers can usually get measurements for an executed program by having a negligible effect on the performance to be measured since most of the time; no other code is being executed. A reasonable sampling period allows the program to execute the same way as usual up to the point an interrupt occurs in order to take a sample. Some inaccuracies in the data measured can still occur since the interrupts can be delayed sometimes, or there can be sections where interrupts are disabled.

The perf tool is based on the events subsystem of the Linux kernel, meaning that it is not limited only to profiling, a number of other features are implemented, for example, it is possible to use the performance counters to measure the number of instructions executed in the course of the execution time of a user provided command. The types of counters we can track are not limited to the hardware counters implemented by the hardware, but we can also count the trace events declared within the Linux kernel (Molnar, 2009) (Motakis, Spyridakis, & Raho, 2013).

In this work this sample based profiling that provides the ability to collect detail information about execution time performance of functions is essential to have accurate data of the hardware performance at function-level, the profiling technique allows us to do so because the processors used by Nokia in its LTE solution are able to record up to six events, as they have six registers reserved for this function. Measuring the revisions as they progress through time with perf allows us to obtain functions-level high-dimensional data, which is required to construct the proposed model by this thesis.

3.2.4 Tracing Technique

One type of data, used to explore the relations from different part of the system, is the event reception and processing times inside the L2 system. This data is collected through the practice of tracing. What is tracing? It can be sometimes not clearly distinguished from other forms of logging or gathering information about the execution of a software, the information gathered is ”low level” and is meant to be used by the development

team, not mainly by system administrators. Let us state a short definition of tracing. Tracing is a powerful analysis technique that consists of collecting information at run time, which provides developers or programmers with several useful data. The most significant advantage of this dynamic analysis is that while running in real-life production environments, it can potentially obtain almost any ”low level” information on program behavior. In exercise, the data collection unsettles the execution of the program, but the perturbation remains reasonably small, potentially insignificant, in several cases with the reasonable use of optimal algorithms (*TracingBook - TracingWiki*, 2009).

The software tracing technique can be an efficient way to record detail information about a program's execution, including event reception and processing times between hardware and software entities. Tracing is suited for applications where the intercommunication of numerous processes with the operating system and with each other is highly complex and where time behavior critical systems could be affected unexpectedly. For example, such online servers with many interactions between several cooperating processes, embedded real-time systems controlling traffic and autos in a Fifth Generation Cellular Network Technology (5G) enabled city or telecommunication systems and its interaction with cellular phones. Tracing has proven to be very efficient in multi-core systems mainly. It is used by developers to face problems whose resolution requires understanding interactions between all layers of a complex system, including third-party products such as hypervisors, OSs, virtual machines, system libraries, and applications. These layers might also be running in different hardware, like general-purpose CPU or DSPs, and be written in completely different programming languages. Much problematic behavior from complex systems only arises when hardware and software interact under real and dynamic workloads; In such situations, a tracer is a tool that can help a great deal to the proper identification of the issue (Toupin, 2011).

Tools for data analysis meet the challenge of staying interactive while managing vast amounts of data. Even within a compact binary form, traces might be as significant as 10 gigabytes and more, this often exceeding the amount of readily available RAM; thus, they can not be loaded into the memory. Whenever a user zooms into a given region of any such trace, this could take some time for all the i/o and calculations required to obtain state information. A way to address these problems may be to read the complete trace once it is opened in order to pre-calculate a number of desirable properties and values.

A large number of different analysis may be performed on a trace. Examples include computing the causality links between events (thus finding the time-critical path between two events), or searching for specific patterns (excessive swapping, spurious timeouts, overloaded disk subsystems).

One crucial step of any data analysis technique is how to visualize the information contained in such data, in this case, of the traces. The preprocessing of the tracing data before its representation can be done with the use of several tools, such as filter, to select the events shown based on their type and the value of their fields. Visualization methods for traces often have event list to easily display a table of events, similar to a database viewer ; control flow view to display the state of various objects, i.e., process, disk, CPU and alike, in function of time, in a way similar to a lot of Gantt graphs; histogram of events, number of events from a specific type, value of a field over a particular type of event, graph showing, for instance, the amount of disk read requests through time; as well as, statistical information like the total number of events by type, average

and duration (*TracingBook - TracingWiki*, 2009).

In this work, tracing provides the ability to collect detail information about event reception and processing times between the different components of the system, this provides accurate data of key relevant processes for the correct execution and fulfillment of function and non functional requirements, the tracing technique allows us to do so because Nokia has implemented custom hooks in the L2 software, and using its own tracing tool, it allows us to obtain event-level high-dimensional data, which is required to construct the proposed model by this thesis.

An excellent visualization technique to understand easily with ease, both profiling and tracing data is flame graphs, for example, such as Figure 9. In this tailor-made graph, a stack trace is displayed in columns of boxes, in which each box represents either a function or an event. The y-axis presents the stack depth information, showing the hierarchy of parent and son events, the top box shows the function or event that was being executed at the moment of taking the sample. The x-axis of flame graphs does not represent time but rather the frequency of meta-functions or events that were found in the stack traces at the moment of sampling, in other words, the wider the box, the most frequent the function or event occurs in the system (Gregg, 2016, p. 7–8).

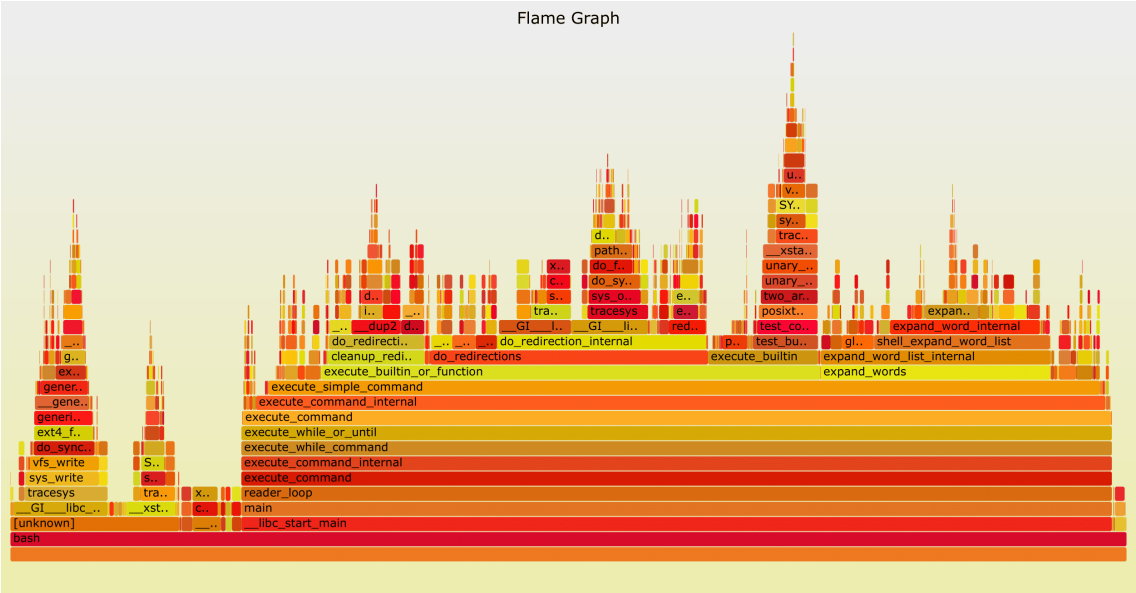

Figure 9: Flame graph example taken from Gregg (2019) stack trace visualizer github repository

3.3 Machine Learning

Machine learning is a broad domain, and it can be divided into three types of learning methods, supervised, unsupervised, and reinforcement, they differ from one another by the nature of their inputs and outputs. In supervised methods the input and output is given by a human, in this sense, the machine is shown the outputs of specific inputs, in order to eventually predict a not given output from a new input (this will be the method used in this thesis work), In unsupervised methods, the machine only is feed with some inputs and its output is used if its satisfactory, and finally, in reinforcement learning, similar to unsupervised learning, the machine only knows some ”human” inputs (pro-

vided by a human) and then generates it's "machine" output, but in this case, as the name suggests, there is a reward or punishment for these generated outputs, with the goal of guidance to the desired solution, neural networks are a typical example of a method that can be used for reinforcement learning (Kojouharov, 2019).

A very straight forward to categorize the most common methods and algorithms is once again in three categories, Basic regression, cluster analysis, and classification. In the basic regression category, there is linear and logistic[3] regression, both methods using supervised learning. In the cluster analysis category, there are many methods; for example, the k means the method and spectral clustering. Both can be implemented with supervised or unsupervised learning. Finally, and the most relevant category for this thesis, the classification (also able to do regression) category contains; all different types of neural networks, K-neighbors, decision trees, random forest, Support Vector Machine (SVM) and Bayesian methods (Kojouharov, 2019).

As the domain of machine learning is too broad and is not so relevant to enter in details for the objective of this thesis, the most critical parts and the most relevant algorithms to be used to create the predictive model will be explored next.

3.3.1 Feature Selection

One crucial part of machine learning in data science is feature selection. Features can be crucial to help a model have good predictive power, and at the same time, if selected poorly they can affect the model's predictive power in a great deal (Chandrashekar & Sahin, 2014, p. 1–2). Feature selection goal is, as its name suggests, to select the optimal subset of variables, reducing noise from irrelevant variables while achieving accurate predictions as output (Guyon & Elisseeff, 2003).

The primary motivation to the study of feature selection and reduction is the fact that the evaluation of all subsets of n variables is of the size of (2^n) and this becomes an NP-hard problem, as the number of features grows, which is often the case (Chandrashekar & Sahin, 2014). One can categorize the feature selection methods broadly into three, filter methods, wrapper methods, and embedded methods.

3.3.1.1 Filter Methods Filter methods use variable rating methods to choose variable by ordering as the main criteria. Rating methods are often used for practical applications due to their easiness and effectiveness. The rating of variables is premised on a suitable rating measure as well as a threshold used to delete variables underneath the threshold (Chandrashekar & Sahin, 2014, p. 17). For example, one of the most straightforward ways to filter irrelevant variables is removing features that do not meet a variance threshold, in other words, variables that do not change much through different samples will most likely be removed, given that they do not provide much information. Standard rating methods are Pearsons correlation coefficient criteria and Mutual information. It is easy to understand by using as an example a Bernoulli random variables variance to decide which features to eliminate,

$$Var[X] = p(1-p) \tag{1}$$

[3]This in contrast to linear regression, the output variable is categorical.

In the Formula 1, one can replace the probability p for the percentage of target variance to keep considering the feature in the model. By choosing $p = 0.7$, in this example, all binary features that have a probability over $0.7(1 - 0.7)$ of being one of the two binary states, will be removed from the data. This could be interpreted in other words as; the feature was many times the same value, ergo, did not provide much information.

The most used filtering method could arguably be the Pearson's correlation criteria used to detect linear dependencies from the input variables towards the output one, in the Formula 2, one can obtain a $R(i)$ value for every input variable x_i by dividing the covariance $cov()$ of the input and output variable, by the square root from the multiplication of the variance $var()$ of the input and output variable, x_i and Y (Chandrashekar & Sahin, 2014, p. 17).

$$R(i) = \frac{\text{cov}\,(x_i, Y)}{\sqrt{\text{var}\,(x_i) * \text{var}(Y)}} \tag{2}$$

Another standard filtering method is the Mutual Information (MI) criteria, measuring variables dependency, using Shannon's Information Theory entropy definition shown in Formula 3.

$$H(Y) = -\sum_{y} p(y)\log(p(y)) \tag{3}$$

This formula represent the uncertainty of the output variable Y.

$$H(Y|X) = -\sum_{x}\sum_{y} p(x,y)\log(p(y|x)) \tag{4}$$

Adding another variable to the system represented by Formula 4 and checking what is its impact towards the uncertainty of Y reveals a delta that can be calculated by Formula 5,

$$I(Y, X) = H(Y) - H(Y|X) \tag{5}$$

This will provide the MI between two features or variables, X, and Y. It will be 0 if the variables are independent and more significant than 0 if they provide information about the other, ergo, having dependency relationship (Chandrashekar & Sahin, 2014, p. 18).

The benefits of filtering methods are that they are computationally fast and prevent overfitting and are demonstrated to operate well for specific datasets. Filter methods do not work on biased learning algorithms that are equal to altering information to suit the learning algorithm. Salient features which are less informative on their own but are informative once paired with others could be thrown away using Pearson's correlation and MI criteria; and also, there is no perfect technique for selecting the size of the feature space (Chandrashekar & Sahin, 2014, p. 18).

3.3.1.2 Wrapper Methods

In comparison to filter methods that use feature-relevance criterion, Wrapper methods operate on classification to obtain a good feature subset, they use the predictor as a black box and predictor performance to assess the variable

subset as the objective function. Because evaluating 2N subsets becomes an NP-hard problem, suboptimal subsets are found using search algorithms that heuristically find a subset (Chandrashekar & Sahin, 2014, p. 18). known wrapper algorithms are: the sequential selection kind, Sequential Feature Selection (SFS) and Sequential Floating Forward Selection (SFFS), and the Heuristics search algorithms such as the Genetic Algorithm (GA) (Goldberg, 1989).

Sequential selection algorithms are from iterative nature, either they start with an empty set adding one by one the features that achieve the highest objective function value, or they can start by having the complete set of variables removing features based on the lowest decrease in the performance of the predictor, such as Sequential Backward Selection (SBS). The disadvantages of sequential selection algorithms is the production of nested subsets, this can happen due to the unconditional forward inclusion, possibly including two high correlated variables. Nevertheless, there have been modifications to the approach to deal with this issues such as the Asaptative Sequential Forward Floating Selection (ASFFS) algorithm, which in theory should produce better results than the common SFFS, but in practice, there is evidence that suggests, these results are dependent on the data distribution and the objective function (Chandrashekar & Sahin, 2014, p. 19–20).

The main disadvantage of Wrapper methods is the number of calculations necessary to achieve a subset of features. The predictor produces a fresh model for each sub-set evaluation, i.e., the predictor is trained for each sub-set and evaluated to extract the accuracy of the classifier. If the amount of samples is vast, much of the execution of the algorithm is spent training the predictor (Chandrashekar & Sahin, 2014, p. 20).

3.3.1.3 Embedded Methods

The embedded methods are combinations of the filter and wrapper methods. These methods have the aim to increase performance in the task of reclassifying multiple feature combination subsets done in wrapper methods, incorporating as part of the training process the feature selection.

In contrast to filter and wrapper approaches, in embedded methods, the learning part and the feature selection part cannot be separated, the structure of the class of functions under consideration plays a crucial role.

Embedded methods comprehend; Forward-Backward Methods, forward with Least Squares and Decision Trees. Backward weight based, Recursive Feature Elimination (RFE). For this thesis is relevant to go deeper to feature selection by embedded methods as an optimization problem. A linear model can be interpreted as a result of the minimization problem represented in Formula 6.

$$\min_{\mathbf{w},b} \frac{1}{m} \sum_{k=1}^{m} L\left(\mathbf{w} \cdot \mathbf{x}_k + b, y_k\right) + C\Omega(\mathbf{w}) \tag{6}$$

In this equation, $L\left(f\left(\mathbf{x}_k\right), y_k\right)$ represents the loss function from $f(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x} + b)$ at the point fo training $(\mathbf{x}_k, y_k)$. And the penalization term $\Omega(\mathbf{w})$ gets an empirical trade-off balancing coefficient C.

The empirical loss can be calculated in different manners, using L^p Lebesgue spaces

metrics[4], for example:

1. The ℓ_1 hinge loss

$$\ell_{1\text{ hinge}}(\mathbf{w}\cdot\mathbf{x}+b,y) := |1-y(\mathbf{w}\cdot\mathbf{x}+b)|_+ \tag{7}$$

2. The ℓ_2 loss

$$\ell_2(\mathbf{w}\cdot\mathbf{x}+b,y) := (\mathbf{w}\cdot\mathbf{x}+b-y)^2 \tag{8}$$

3. The Logistic loss

$$\ell_{\text{Logistic}}(\mathbf{w}\cdot\mathbf{x}+b,y) := \log\left(1+e^{-y(\mathbf{w}\cdot\mathbf{x}+b)}\right) \tag{9}$$

Likewise, the penalizing them $\Omega(\mathbf{w})$ from Formula 6 can be understood, for example, as:

1. The ℓ_0 norm

$$\Omega(\mathbf{w}) = \ell_0(\mathbf{w}) \tag{10}$$

   the function $\ell_0(\mathbf{w})$ contains the number of non-zero coordinates of $\mathbf{w}$.

2. The ℓ_1 norm

$$\Omega(\mathbf{w}) = \sum_{i=1}^{n} |w_i| \tag{11}$$

In this thesis, one important method used for feature selection is the embedded method Least Absolute Shrinkage and Selection Operator (LASSO). LASSO technique tries to solve the minimization problem represented by Formula 12, who uses the ℓ_2 loss function and a constraint on the parameters of the ℓ_1 norm.

$$\min_{\mathbf{w},b} \sum_{k=1}^{m} (\mathbf{w}\cdot\mathbf{x}_k - y_k)^2 \tag{12}$$

This method being subject to $\sum_{i=1}^{n} |w_i| \leq \sigma_0$, and thus, the use of this ℓ_1 norm constraint produces a sparse model (Lal, Chapelle, Weston, & Elisseeff, 2006, p. 14–17). This method gave the best results regarding feature selection, allegedly because it produces a sparse model, and moreover, it has been known empirically to be very effective.

3.3.2 Regression

Supervised machine learning algorithms, i.e., using parts of known datasets as training data, can solve either a regression or classification tasks, the difference between these two tasks relies on the estimation from the predicted variable; in regression the output

[4] L^p spaces are an important class of Banach spaces in functional analysis, they are used in many fields due to their key role on analysis of measure and probability spaces.

variable is numerical or continuous, in comparison to a classification task where the output variable is categorical. Having this in mind, in this thesis work, the output variables, the PM are continuous, so the algorithms explored solve regression tasks.

Various techniques can do regression, it can be done by SVM, by neural networks, regression trees, gradient boosting, k-nearest neighbors algorithm and many others. it is also possible by ”simple” linear regression that consist of a simple model where given the point of an independent variable and one dependent variable, in other words, dots in a plane, the task of regression is to find a straight line where the total vertical distance from all the dots in the plane to the straight line is the lowest possible, meaning that with this line, one could predict the value of the dependent variable value of a given new value for the independent variable. This being a straightforward and basic explanation of regression, for this thesis it is relevant to explain the LASSO method for regression, this because LASSO method reveals handy information about the estimators and its influences on best subset selection (Tibshirani, 1996; Kim, Koh, Lustig, Boyd, & Gorinevsky, 2007; Friedman, Hastie, & Tibshirani, 2010), especially relevant for this thesis.

3.3.3 Clustering

The two algorithms mentioned in the introduction to the machine learning background were K-means and spectral clustering. The K-means algorithm works by creating clusters of data separating samples in a given n number of groups of similar variance; it can be seen as solving the minimization problem of sum of squares between the clusters. The algorithm is known to scale appropriately to a large number of samples, and it has been used in many domains (Arthur & Vassilvitskii, 2007).

Spectral clustering produces a low-dimension embedding of the affinity matrix among samples, accompanied by a K-means in the low dimensional space. It is especially useful if the affinity matrix is sparse. It works properly for a small number of clusters but is not advised when handling many clusters (von Luxburg, Belkin, & Bousquet, 2008).

3.3.4 Neural Networks

The concept of neural networks uses various analogies from biological neural networks, that is where they got the name from, in reality, a neural network refers to any layered connected structure of networks, They are very popular on the machine learning field because they have proven to be effective in various tasks. The distinction between a ”normal” neural network and a deep neural network relies on the number of hidden layers.

What is a neural network?, it starts by the fundamental block called perceptrons (the analogy of a neuron), when connected these compose hierarchical models also known as neural networks as seen in Figure 10. The key to the learning in neural networks is backpropagation (LeCun, Bottou, Orr, & Müller, 2012; E. Rumelhart, E. Hinton, & J. Williams, 1986), and gradient descend techniques, where the internal states of the neurons are punished or rewarded (by supervised learning) propagating from the output layer and through all the hidden layers, in this way updating and approaching their values to the desired output of the model.

In this thesis work, the Multi-Layer Perceptron (MLP), a neural network with many layers, and with back-propagation provide outstanding results. The MLP is a supervised learning algorithm that learns a function by training on a dataset. Given a set of features and a target, it can learn a non-linear function approximator for regression. It is characterized for having one or more non-linear layers between the input and the output layer, called hidden layers.

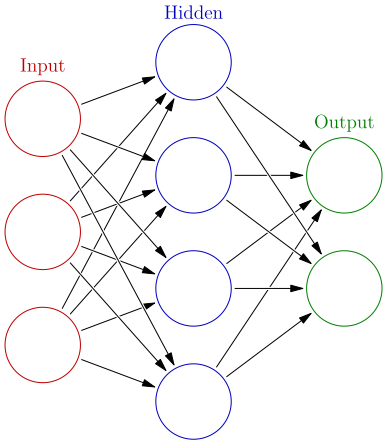

Figure 10: Classic representation of a neural network with one hidden layer,

In neural networks and even more in deep learning, there is the problem of over-fitting. What are under-fitting and over-fitting? As their names suggest when a network is trained ”too much” it may risk just memorizing the inputs and not being able to predict the output of new given input accurately. So under-fitting is the opposite situation, where the network does not have enough information to predict accurately. Techniques as adaptive learning rates, batching, and regularization is used to prevent overfitting.

Sequence modeling creates models based on samples that have multiple data points, for example, the PM of a specific revision in a software project, a more general example could be the problem of self-parking car, the instructions to achieve the desired outcome need to be modeled based on all the combination of previous movements. Sequence Modeling may be relevant to our quantitative data approach because it is meant to model a data set of many examples with the particularity that each example has multiple data points assuming that these data points interact with each other in complex ways, This could be the case if one assumes that the past states of the system influence the present and the future of itself, which, after some experiments and reflection, may not be the most accurate way to understand the LTE L2 software system, it is correct that its components are highly related, but the measurement of one point in time probably doesn’t affect the next measurement.

3.4 Visualization technology

The visualization of multivariate data is a complete ongoing field of research counting on different applications, luckily, there exist environments that facilitates and simplifies the data science work by a great extent, in this thesis, all code is done in python to processes the raw data, as well as for the data exploration and graphical representation

of the results, the later is done in *jupyter notebook*[5], the web-based interactive computational environment, using Python as programming language. this choice reflects on the flexibility of re-running individual code snippets, which allows agile testing of ideas, (i.e., new graphs ideas, models, feature selection algorithms, etc.)

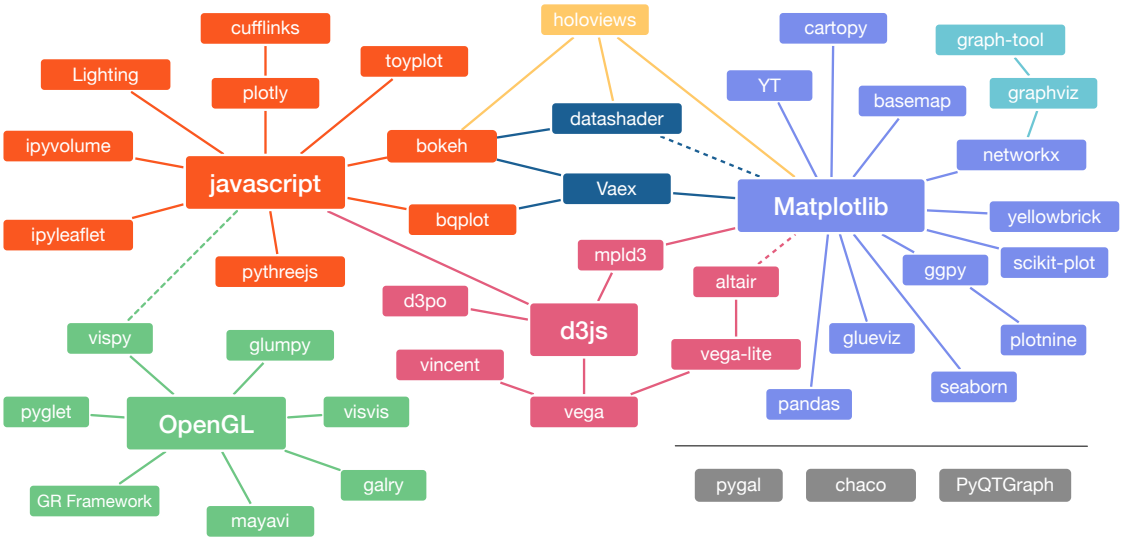

Figure 11: adaptation the Python Visualization Landscape slide from VanderPlas (2017), taken from (Rougier, 2019).

As python is the primary programing language, the task of choosing the most adapted visualization technology will be determined by coupling level from other essential libraries used for handling the data. The excellent landscape overview of the coupling level from python visualization related libraries done by VanderPlas (2017) can be seen in Figure 11. We can observe how libraries work together and how close they are related, taking into account that *Pandas*[6] is the main library choice to handle the data processing and transformation, the natural starting point to explore would be all libraries related to *matplotlib*.

There is a sample of a scatterplot matrix done with both libraries, the one done with *seaborn* library in Figure 12 and the one done with *plotly* in Figure 13. The decision relays on balance between more interactivity and easy usage from *plotly* compared to a more stable, well known and mature library as *matplotlib*

[5]https://jupyter.org/about
[6]https://pandas.pydata.org/

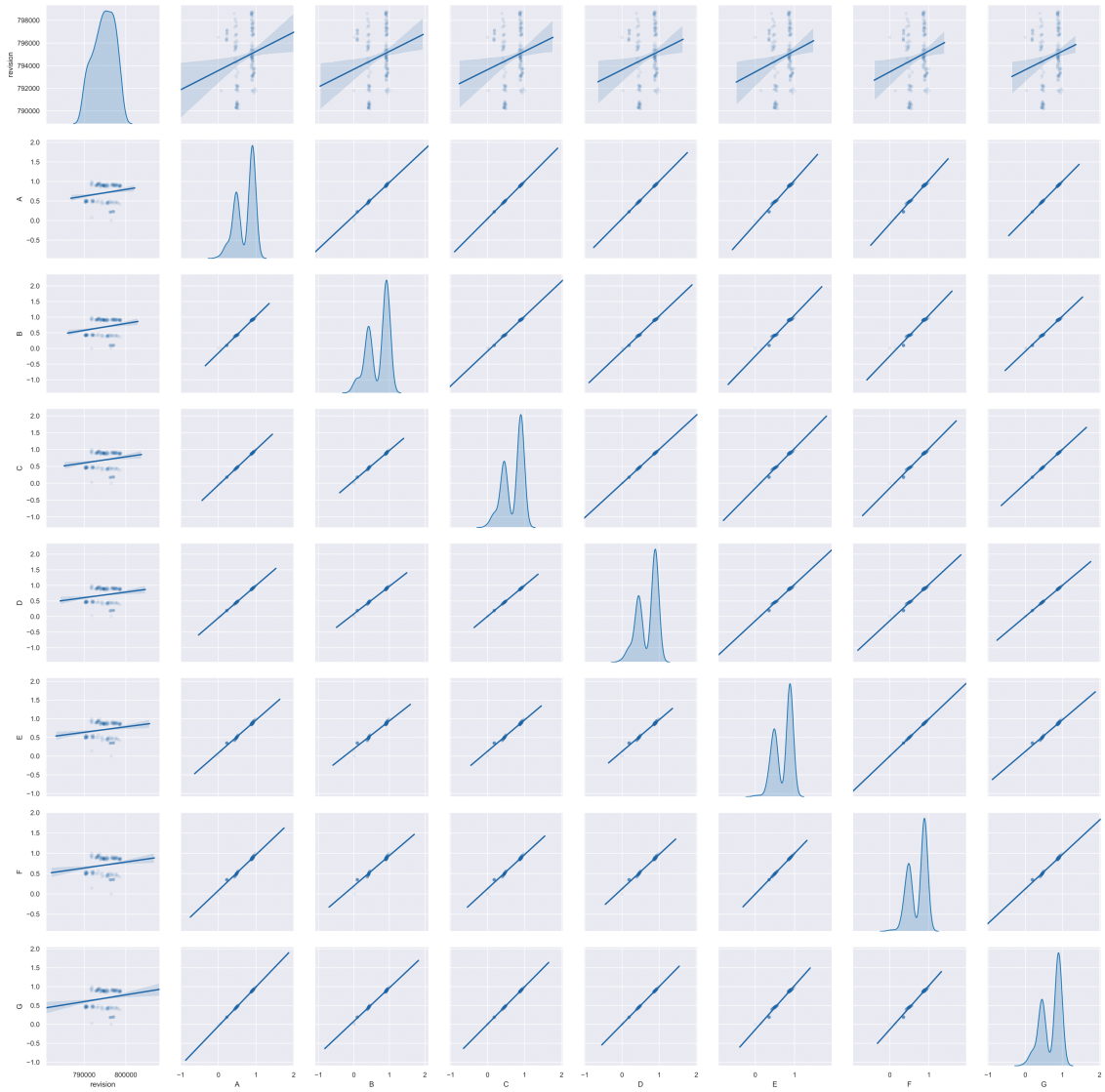

Figure 12: Example of a scatterplot matrix with regression line using seaborn

The main library in which the visualizations are presented is *seaborn*[7], this decision was made by the following reasons:

- It is based on the well known, fast, and well documented 2D plotting library *matplotlib*[8].
- The high coupling it was with *Pandas*, which is the main library used to handle the data
- It is user-friendly and with very few code (in comparison to for example *bokeh*), it creates very aesthetic appealing graphs.
- It contains a vast functionality to create multiple graphs, and it is simple and very well documented.

[7]https://seaborn.pydata.org/
[8]https://matplotlib.org/

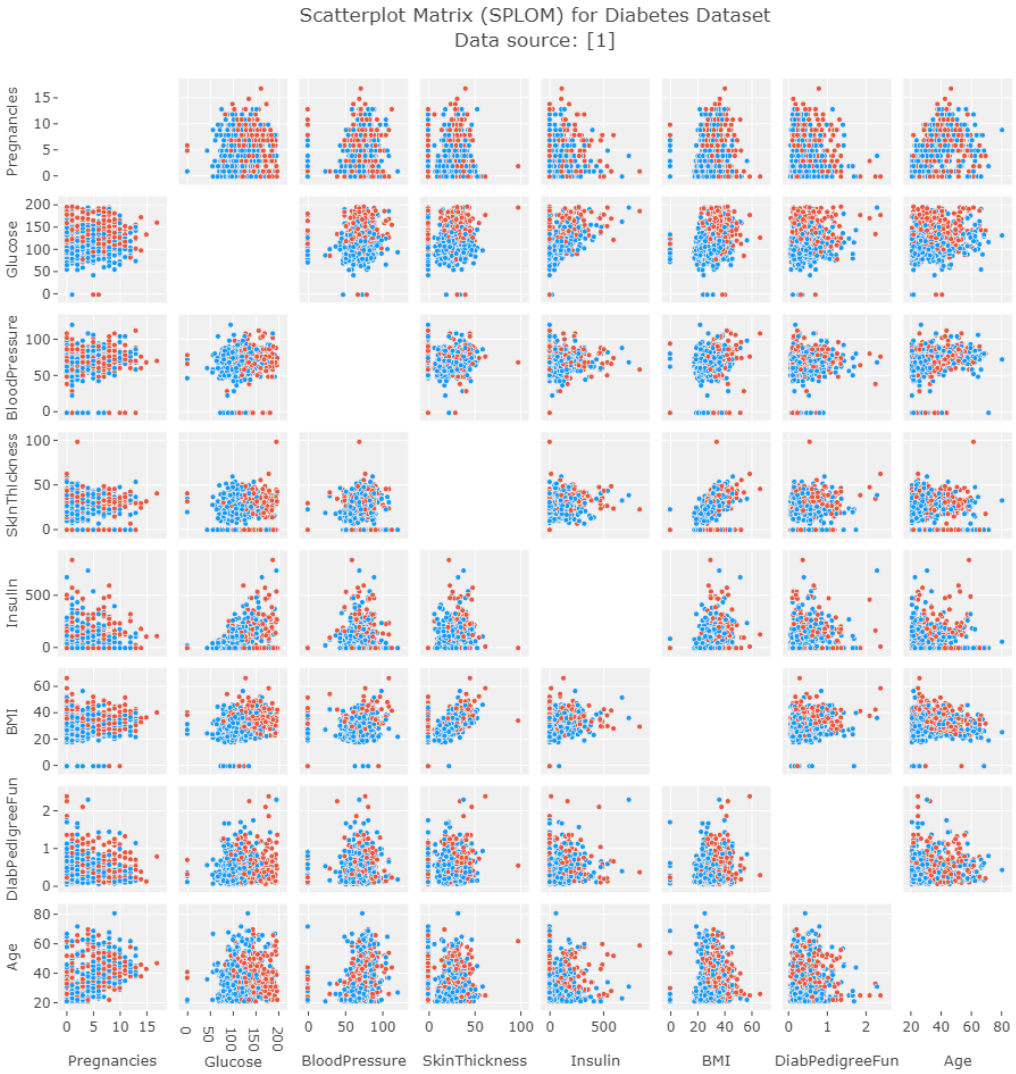

Figure 13: Example of a scatterplot matrix using plotly taken from (plot.ly, n.d.).

The drawbacks or compromises from this choice, are in interactivity possibilities from the graphs, it is clear, that new promising libraries as *plotly*[9] can create interactive graphs with great flexibility and simplicity, it can be to a great extent much slower than *matplotlib* or *bokeh*. Also with the current choice of *matploblib* based library *seaborn*, it is possible to add interactivity with some effort using *mpld3*, which is a comprehensible compromise for a library that is fast and easy to use.

[9] https://plot.ly/

4 Model Creation Process

Following the DSRP, the second activity addresses the question *What would a better artifact accomplish?*. In the case of this research, the question to address is: What would the improved predictive model accomplish better than the other various testing protocols and tools Nokia has today running at the Research and Development (R&D) level?

The expected benefits of this new artifact are a better estimation of the affectation regarding DL throughput of the system before committing a new change of code to the production branch. The evaluation of this better estimation is not to be compared, yet it would provide another estimation on top of the ones already inferable by all the testing practices are currently done at the R&D process.

The overall design for constituting the proposed approach can be seen in Figure 14, the process starts by the data recollection, the data pre-processing, the exploratory analysis and finally the model creation, each of these steps will be explained in detail further on. The process is designed to allow a continuous integration of a new selection of most relevant data at the pre-processing step; it also needs to be flexible to the fluctuation in the availability from some tools in a development environment.

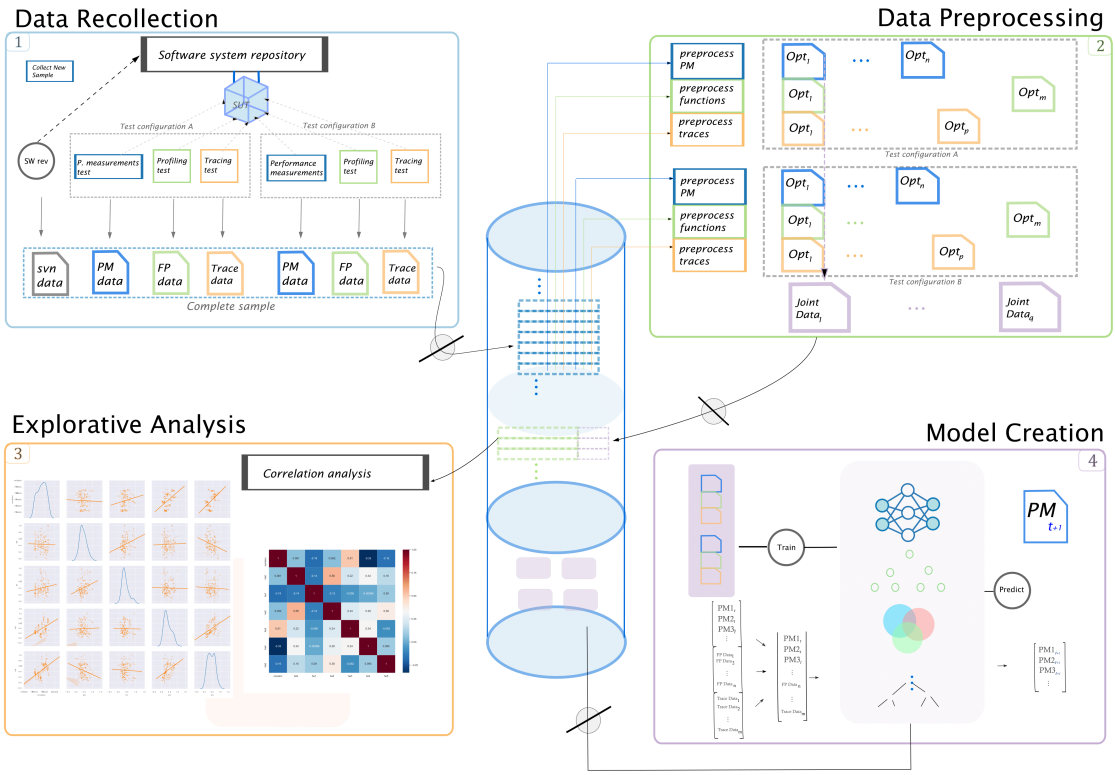

Figure 14: Predictive model creation process approach, overall architecture.

The architecture represented in Figure 14 enables a continuous modeling and analysis of different system properties, and is it the current outcome of the **Cycle 2** (2.3) of this DSR process.

4.1 Data Recollection

The data collected using the existing environment for SCT is to be used by gathering details such as the SW release, the revision, the information about the specific software and hardware configuration, in order to eventually create a data trend.

The preparation of the data is essential prior to its analysis, The data used for in this work comes from three different levels of abstraction, all related by coming from the same SUT; The first level data comes out of the tests results from the throughput related measurement indicators; The second level data belongs to the performance test results at individual function-level performance in the L2 software obtained by profiling using the perf tool. Moreover, the third level data is gathered from the results of the tracing test; this being event-level data contains information about resources and time used during the event calls.

There is a need to pre-process the relevant data from the test results and to prepare it for the desired analysis. To do the prepossessing, Python with pandas and Hierarchical Data Format 5 (hdf5) library was used to processes and filter fast and easily the data and to compress the results of the pre-processed data to maintain a controlled size from all the data collection process.

4.1.1 Data-flow Process

The processes of collection the quantified data from the three different layers is done by running three times each of the two test suites selected for the study, having a total of six test runs per the software revision. The script used for the axiomatization of the test runs was designed to obtain as many data as possible. Also, it is executed continuously as soon as a new software revision is available. The decision of gathering the test results by three separate test runs, using on each of those runs specially designed tools, was because such tools are also part of the development processes. Thus they are likely to fail for short periods, this being normal for systems under development. Taking into account this possible intermittent availability of the tools, the separation of the test runs will permit that in some cases at least the partial recollection from the full six tests samples.

This prompt a need to deal with absences of some system layer's data samples, such revision samples that are not complete then will not be used for the model creation, but will indeed enrich the exploratory analysis of the pre-processed data.

The outcome of **Cycle 1** (2.3) of this DSR process is represented in Figure 15. There are two different test scenarios where the data is collected, these were selected to compare different hardware design, these two test configurations work with a multiple eNodeB hardware architecture designs, and they differ in the number of eNodeB pools, as well as the objectives of the test. They were chosen as a starting point for this thesis work, and to compare the difference between the samples from one another.

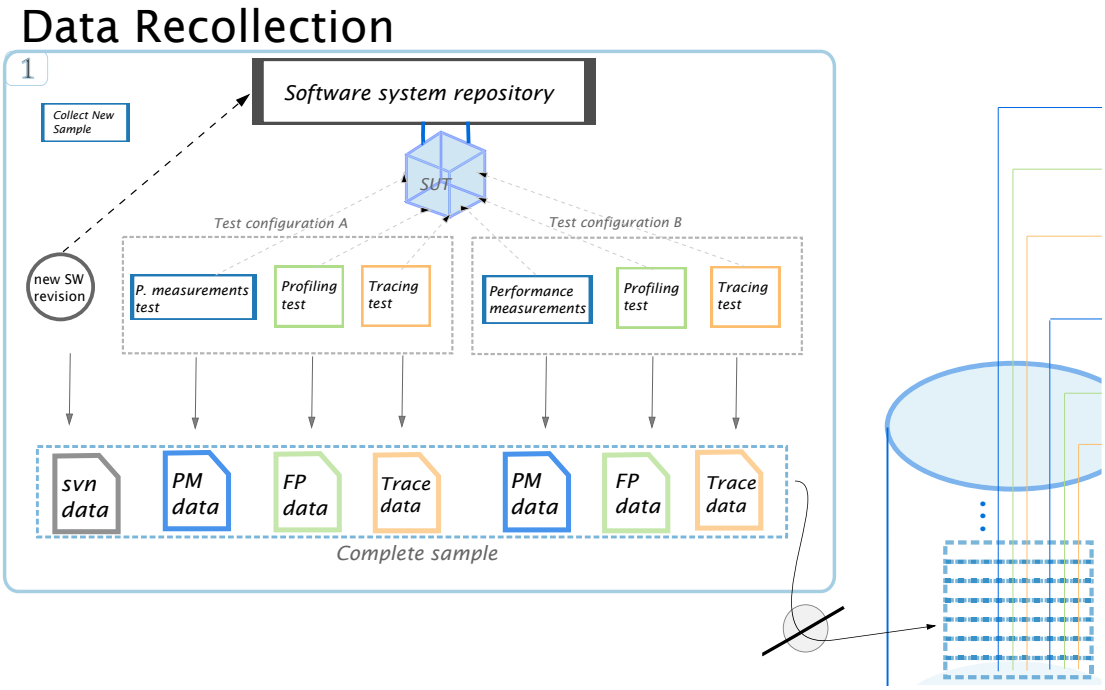

Figure 15: First Step, Continuous Data Collection.

Each entity in Figure 15 is color mapped to each layer of the system, following the layered benchmark framework from Muller (2007); blue color is assigned to PM that represent the service layer of the system, green color is assigned to the function-level profiling data, that represents the hardware layer of the system, and finally the orange color is assigned to the event-level tracing data that for this works represents the operative system or middleware layer of the system.

This step recollects all the raw data coming from the test runs, saving them by date in a database, and ready to be fetched in the data preprocessing step to generate a dataset from different features joined by revision number.

4.1.2 Automating Data Collection

In order to have better predictive power, a model requires more historical data points, meaning the proper collection of various tests results during the evolution of the development needs to be efficient in the number of tests runs. An always ongoing script checks updated the software repository looking for a newer revision as the one from the last recollection data. This goes with the principle of CI, thus creating an automated continuous new revision data collection.

4.2 Data Pre-processing and Filtering

The organization from the different preprocessing options can be appreciated in Figure 16, after the raw data is stored in a joint database, the pre-processing data functions can access and transform the data, always maintaining the raw data, this at least during the exploration for the best data transformation. As seen in Figure 16 for each layer: down-

link performance measurements, function-level profiling data, and event-level tracing data. Different pre-processed data outputs are generated and equally stored, in order to be used by a continuous exploratory analysis or to be joint to start a model creation instance.

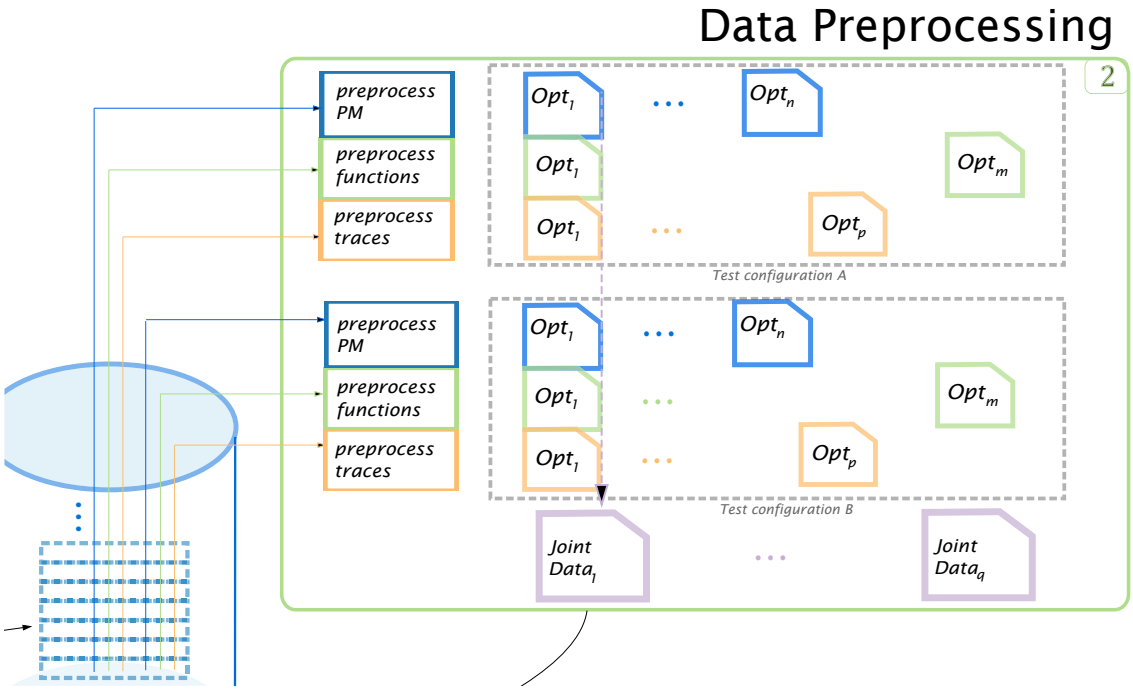

Figure 16: Second Step, Continuous Data Pre-processing.

This organization of the data allows for the maximum flexibility regarding missing revision samples, or eventual errors in the sample taking test executions. This way, the data from any of the three layers, even if the other data is missing or corrupted, can be used to the exploratory analysis with linear regression, to be done for each layer.

First, the joining process from all the samples horizontally by the revision is automated to start having some descriptive statistic from the layer divided dataset, that will take part in the exploratory analysis.

In this step, the three preprocessing options were implemented for the function-level data, two for the event-level traces, and only one the down-link throughput PM.

For the down-link throughput PM data the following pre-processing was done by joining all the 13 PM values by PM names, taking into account that these thirteen PM measure related down-link throughput performance characteristics multiple times during the test run. The outcome dataset could be represented as the following matrix; pm meaning performance measurement, and n the number of different revision from the sampled function-level data:

$$P = \begin{bmatrix} pm_{0,0} & pm_{0,1} & \dots & pm_{0,12} \\ pm_{1,0} & pm_{1,1} & \dots & pm_{1,12} \\ \vdots & & & \\ pm_{n,0} & pm_{n,1} & \dots & pm_{n,12} \end{bmatrix}$$

For the function-level data the following pre-processing was done:

4.2.1 Function-level Pre-processing Options

the first preprocessing option was to join all the functions by HPC events, these were six different events that the hardware allows us to record, so the outcome data could be represented as the following matrix; he meaning hardware event, and n the number of different revision from the sampled function-level data:

$$F = \begin{bmatrix} he_{0,0} & he_{0,1} & he_{0,2} & he_{0,3} & he_{0,4} & he_{0,5} \\ he_{1,0} & he_{1,1} & he_{1,2} & he_{1,3} & he_{1,4} & he_{1,5} \\ \vdots & & & & & \\ he_{n,0} & he_{n,1} & he_{n,2} & he_{n,3} & he_{n,4} & he_{n,5} \end{bmatrix}$$

the second preprocessing option was made by joining all functions by hardware event and core, into one value, in other words the columns from hardware events and core were dropped, and then the common function names values were added up, so the outcome data could be represented as the following matrix; f meaning function, n the number of different revision from the sampled function-level data, and m the number of common functions through all revision samples:

$$F2 = \begin{bmatrix} f_{0,0} & f_{0,1} & \dots & f_{0,m} \\ f_{1,0} & f_{1,1} & \dots & f_{1,m} \\ \vdots & \ddots & & \\ f_{n,0} & f_{n,1} & \dots & f_{n,m} \end{bmatrix}$$

the third preprocessing option was made by letting values be combinations of functions name, hardware event and core, then only the combination values that existed in all revision samples were joined together so the outcome data could be represented as the following matrix; fhe meaning function hardware event and core, n the number of different revision from the sampled function-level data, and p the number of common function hardware event and core through all revision samples:

$$F3 = \begin{bmatrix} fhe_{0,0} & fhe_{0,1} & \dots & fhe_{0,p} \\ fhe_{1,0} & fhe_{1,1} & \dots & fhe_{1,p} \\ \vdots & \ddots & & \\ fhe_{n,0} & fhe_{n,1} & \dots & fhe_{n,p} \end{bmatrix}$$

It is important to state that F is smaller than $F2$, $F2$ being smaller than $F3$, also that $m << p$, in other words, the more high dimensional data is generated by preprocessing option three and is represented by $F2$ matrix.

4.2.2 Event-level Pre-processing Options

For the event-level data the following pre-processing was done:

the first preprocessing option was to filter four events that had been marked as of interest due to experts advise, the outcome dataset could be represented as the following matrix;

se special event, and n the number of different revision from the sampled event-level data:

$$E = \begin{bmatrix} se_{0,0} & se_{0,1} & se_{0,2} & se_{0,3} \\ s_{1,0} & se_{1,1} & se_{1,2} & se_{1,3} \\ \vdots & & & \\ se_{n,0} & se_{n,1} & se_{n,2} & se_{n,3} \end{bmatrix}$$

the second preprocessing option is similar to the second function-level pre-processing option, it is done by joining all event times by event name, then only the event names that are common to all revision samples remained in the dataset so the outcome data could be represented as the following matrix; e meaning event, n the number of different revision from the sampled event-level data, and m the number of common functions through all revision samples:

$$E2 = \begin{bmatrix} e_{0,0} & e_{0,1} & \dots & e_{0,m} \\ e_{1,0} & e_{1,1} & \dots & e_{1,m} \\ \vdots & \ddots & & \\ e_{n,0} & e_{n,1} & \dots & e_{n,m} \end{bmatrix}$$

Again it is good to state that matrix E smaller $E2$, so the more high dimensional data is generated by preprocessing option two.

The creation of different pre-processing options is done iteratively and are evaluated to previous options to assess their usefulness, this in the style of DSR. These are the ones created at the moment of the writing of this thesis.

As one can observe in Figure 16, that all the preprocessing option can be combined to construct a predictive model that will help asses which combination of options gives the better results, this implies 11 possible combinations at the moment:

- $P + E$
- $P + E2$
- $P + F$
- $P + F2$
- $P + F3$
- $P + F + E$
- $P + F2 + E$
- $P + F3 + E$
- $P + F2 + E2$
- $P + F3 + E2$

These different pre-processing options from raw data and its combinations are the current outcome from the **Cycle 3** (2.3) from this DSR process.

In the next part of the chapter, Only details from the data pre-proceed in the combination **P+F+E** will be presented, this due to the low dimensionality of the data on this pre-processing options, allowing a more accessible presentation of the datasets.

4.2.3 Down-link Throughput Performance Measurements Data

The PM coming from two different test configurations chosen by experts, these measurement record key properties emerging at the different context where the down-link throughput is affected. It is important to that these PM are meet as they are defined in the requirements from the 3GPP (3GPP 36.425, 2019).

As one can see in Figure 15 the PM results are collected from the SUT in the test environment, in the two different test configurations, always in the same revision or state of the system.

After the data collection and "vertical union" we can observe basic descriptive analytics from the PM from test configuration A in Table 1. likewise the basic descriptive statistics from test configuration are found in Table 2.

Table 1: Descriptive statistics from the test A PM dataset.

	A	B	C	D	E	F	G	H	I	J	K	L	M
count	2.01e+02	2.01e+02	2.01e+02	2.01e+02	2.01e+02	2.01e+02	2.01e+02	2.01e+02	2.01e+02	2.01e+02	2.01e+02	2.01e+02	2.01e+02
mean	3.15e+10	2.46e+07	3.27e+06	4.84e+09	7.05e+09	7.09e+09	4.87e+06	3.78e+07	3.78e+07	8.52e+06	1.10e+06	4.68e+06	7.09e+09
std	2.83e+09	3.86e+06	2.93e+05	4.09e+08	8.42e+08	8.46e+08	4.12e+05	3.20e+06	3.20e+06	7.15e+05	9.34e+04	3.95e+05	8.46e+08
min	2.34e+10	1.55e+07	2.50e+06	3.74e+09	4.33e+09	4.36e+09	3.77e+06	2.91e+07	2.91e+07	6.66e+06	8.66e+05	3.66e+06	4.36e+09
25%	2.89e+10	2.10e+07	3.00e+06	4.46e+09	6.24e+09	6.26e+09	4.49e+06	3.48e+07	3.48e+07	7.86e+06	1.02e+06	4.32e+06	6.26e+09
50%	3.35e+10	2.74e+07	3.48e+06	5.13e+09	7.61e+09	7.65e+09	5.16e+06	4.00e+07	4.00e+07	9.02e+06	1.17e+06	4.96e+06	7.65e+09
75%	3.38e+10	2.77e+07	3.51e+06	5.17e+09	7.74e+09	7.77e+09	5.21e+06	4.04e+07	4.04e+07	9.10e+06	1.18e+06	5.01e+06	7.77e+09
max	3.49e+10	2.88e+07	3.62e+06	5.33e+09	8.18e+09	8.22e+09	5.37e+06	4.17e+07	4.17e+07	9.39e+06	1.22e+06	5.16e+06	8.22e+09

In test configuration A there are 201 samples, meaning 201 different states in time where new code has been added and modified the performance of the DL throughput. We can observe the values are in different scales of magnitude.

Table 2: Descriptive statistics from the test B PM dataset.

	A	B	C	D	E	F	G	H	I	J	K	L	M
count	1.92e+02	1.92e+02	1.92e+02	1.92e+02	1.92e+02	1.92e+02	1.92e+02	1.92e+02	1.92e+02	1.92e+02	1.92e+02	1.92e+02	1.92e+02
mean	5.39e+10	1.54e+06	7.86e+05	9.17e+09	1.51e+06	3.09e+07	9.20e+06	7.16e+07	7.16e+07	4.49e+06	1.75e+06	7.03e+06	3.09e+07
std	1.64e+08	1.16e+04	3.22e+03	2.82e+07	6.22e+03	8.76e+04	2.83e+04	2.20e+05	2.20e+05	1.35e+04	5.40e+03	2.16e+04	8.76e+04
min	5.35e+10	1.52e+06	7.78e+05	9.10e+09	1.49e+06	3.07e+07	9.14e+06	7.11e+07	7.11e+07	4.45e+06	1.74e+06	6.98e+06	3.07e+07
25%	5.37e+10	1.53e+06	7.83e+05	9.14e+09	1.50e+06	3.08e+07	9.18e+06	7.14e+07	7.14e+07	4.47e+06	1.74e+06	7.01e+06	3.08e+07
50%	5.39e+10	1.53e+06	7.86e+05	9.16e+09	1.51e+06	3.09e+07	9.19e+06	7.15e+07	7.15e+07	4.48e+06	1.75e+06	7.02e+06	3.09e+07
75%	5.40e+10	1.55e+06	7.88e+05	9.18e+09	1.51e+06	3.10e+07	9.22e+06	7.17e+07	7.17e+07	4.50e+06	1.75e+06	7.04e+06	3.10e+07
max	5.44e+10	1.56e+06	7.95e+05	9.25e+09	1.52e+06	3.12e+07	9.28e+06	7.22e+07	7.22e+07	4.53e+06	1.76e+06	7.09e+06	3.12e+07

In test configuration B, there are 192 samples, meaning 192 different states in time where new code has been added and modified the performance of the DL throughput. We can also observe the values are in different scales of magnitude.

The number of samples between both test configurations is different due to minor unavailabilities and failing states from the test during the developing process.

4.2.4 Function-level data

The profiling of the CPU performance by the individual functions from the source code is done using the perf. The results from the hardware events are counting how many times these six registers in the CPU were overflowed and as a result, triggering an event to be collected as a sample. This gives us information about how many times an event was triggered over a period of time. The recorded events may have different costs in time, for example, it is known that the L2 cache refill level can take up to ten times more time than the L1 cache refill event.

There are six events registered who measure all functions that overflow any of the assigned registers to the particular event on any of the processor's cores. This is recorded by perf and used to make an exploratory analysis and the predictive model creation.

This six types of cache refills and memory reads were selected to be used in this study are based on an expert evaluation of the behavior of the inspected system, who concluded that these six HPC are likely to be the best indicators of the overall system performance. The selection of the appropriate HPC are system-specific, and no single set of HPC could be generalized to have useful indicators of meaningful performance change in all systems (Laivamaa, 2019, p. 45).

Table 3: Descriptive statistics from the test A low-dimensional function-level dataset, pre-processing option 1, matrix F.

	he0	he1	he2	he3	he4	he5
count	223.000000	223.000000	223.000000	223.000000	223.000000	223.000000
mean	2400.511211	6990.502242	9669.461883	844383.053812	1678.573991	133510.183857
std	128.126446	348.543364	540.127277	11278.703736	85.052144	13401.535269
min	1982.000000	5834.000000	8079.000000	821385.000000	1194.000000	103775.000000
25%	2322.000000	6919.500000	9736.000000	836966.000000	1694.000000	128349.500000
50%	2427.000000	7075.000000	9821.000000	840033.000000	1703.000000	128608.000000
75%	2501.500000	7208.000000	9876.500000	846921.500000	1717.000000	128895.000000
max	2632.000000	7492.000000	10276.000000	880898.000000	1766.000000	177609.000000

Table 4: Descriptive statistics from the test B low-dimensional function-level dataset, pre-processing option 1, matrix F.

	he0	he1	he2	he3	he4	he5
count	179.000000	179.000000	179.000000	179.000000	179.000000	179.000000
mean	1912.441341	4646.134078	6739.256983	785195.592179	582.178771	81085.586592
std	89.808352	190.199345	197.915090	3304.728044	20.759495	1711.098703
min	1204.000000	2833.000000	4502.000000	771373.000000	311.000000	58435.000000
25%	1852.000000	4562.000000	6685.000000	782928.500000	581.000000	81104.500000
50%	1923.000000	4644.000000	6714.000000	784369.000000	584.000000	81218.000000
75%	1971.500000	4747.000000	6758.500000	787642.000000	586.000000	81317.000000
max	2055.000000	5161.000000	7024.000000	794325.000000	597.000000	81688.000000

223 samples for test configuration A were collected, basic descriptive stats are found in Table 3. likewise 179 samples for test configuration B where collected, see Table 4. Looking at these statistics we can start to observe that in test A all the mean values from the he are greater than in test B, but in different proportions, $he3$ measure is barely greater, in contrast to $he4$, where test A measures more than three times greater than

test B, also its standard deviation indicates that in test A this value has four times more variability.

4.2.5 Event-level data

For every state of the system or in this case for every revision taken into account for the study, event-level quantified data coming from the tracing tools, is collected from both test configuration A and test configuration B, descriptive statistics in Table 5 and 6.

Table 5: Descriptive statistics from the test A event tracing dataset, pre-processing option 1, matrix E.

	e0	e1	e2	e3
count	203.000000	203.000000	203.000000	203.000000
mean	76.744483	1386.062158	81.641128	184.424291
std	14.573080	367.564016	13.704333	51.838818
min	28.809000	294.625000	39.738000	34.867000
25%	78.131000	1447.381000	83.859500	200.859500
50%	81.853000	1499.481000	85.635000	201.368000
75%	82.932000	1518.996000	88.712000	203.898500
max	91.094000	1785.410000	91.412000	211.855000

Table 6: Descriptive statistics from the test B event tracing dataset, pre-processing option 1, matrix E.

	e0	e1	e2	e3
count	174.000000	174.000000	174.000000	174.000000
mean	63.542218	1239.605575	63.422845	194.376075
std	13.615645	257.278557	15.212970	37.704427
min	20.427000	274.187000	33.625000	25.587000
25%	57.559750	1155.819500	55.651000	200.430750
50%	60.936000	1222.845000	56.993000	201.349000
75%	69.366250	1470.426000	60.957000	206.567750
max	82.758000	1505.817000	89.413000	216.757000

For test configuration A 203 samples and 174 samples for test configuration B, in contrast to the function-level, the test configuration B have in general lower values for all the four analyzed events.

4.3 Exploratory Analysis

To start the exploratory process towards a relevant solution and a contribution to the domain of quantified software engineering, the visual representation of the data coming from the three different system layers, the hardware, service and applications layer data and the possible relations between both function level and requirements level test results, is to be done using jupyter notebook (as it allows easy sketching). As is shown in Figure 17 the exploratory analysis notebooks (this means the code and the functions running

on jupyter notebook) have continuous access to the preprocessed data, coming from the previous step of the architecture, allowing to generate new correlation analysis with the aim of unveiling hidden relations in-between layers.

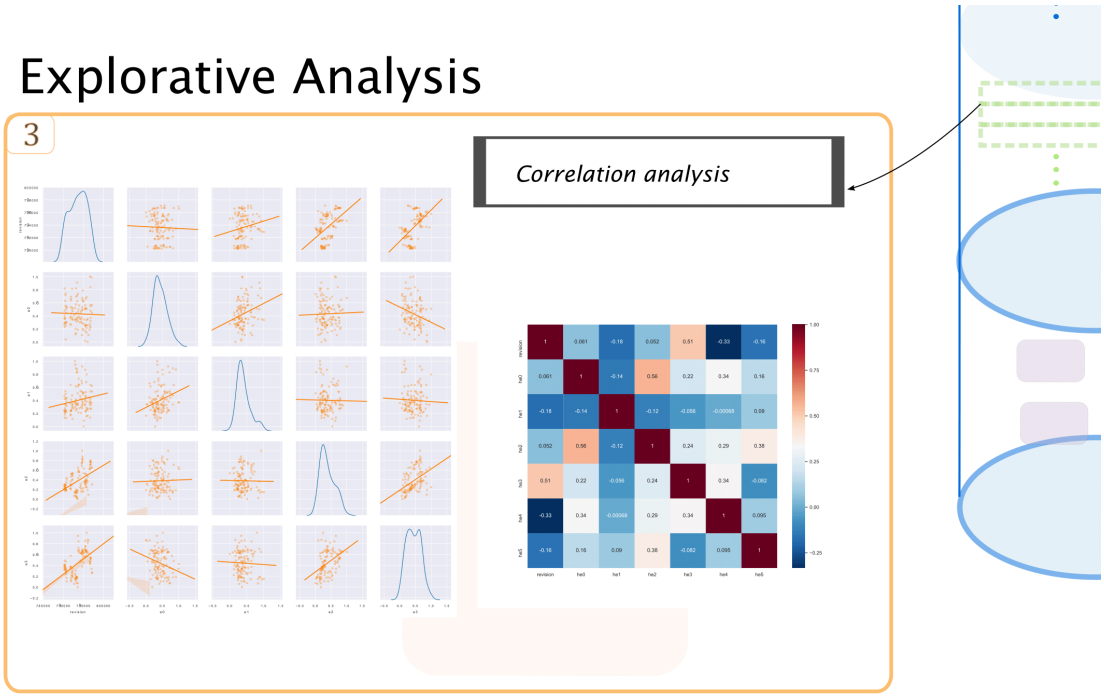

Figure 17: Third step, Continuous Exploratory Analysis.

The multi-variable scatter plot is used to see and observe correlations between multiple variables easily, it is preferred over a correlation heat-map because it also allows us to see clustering information as well as it allows to draw a simple linear regression line, all this information packed in a singular visualization provide a good dense amount of information to analyze, in these case between PMs, cycles used by functions, etc.

This is the current outcome of **Cycle 4** (2.3) from this DSR process. that allows the exploration of inside relations from the different layer's data, results from this step of the overall process can be found in Section 5.1.

4.4 Model Creation

As seen in Figure 18 following the process of joining the three preprocessed data, into one complete frame to start the model creation part of the overall process. The idea is to compare the recall and accuracy of the models to update the preprocessing options, or to perform new transformations that following the DSRP allows the artifact to be improved.

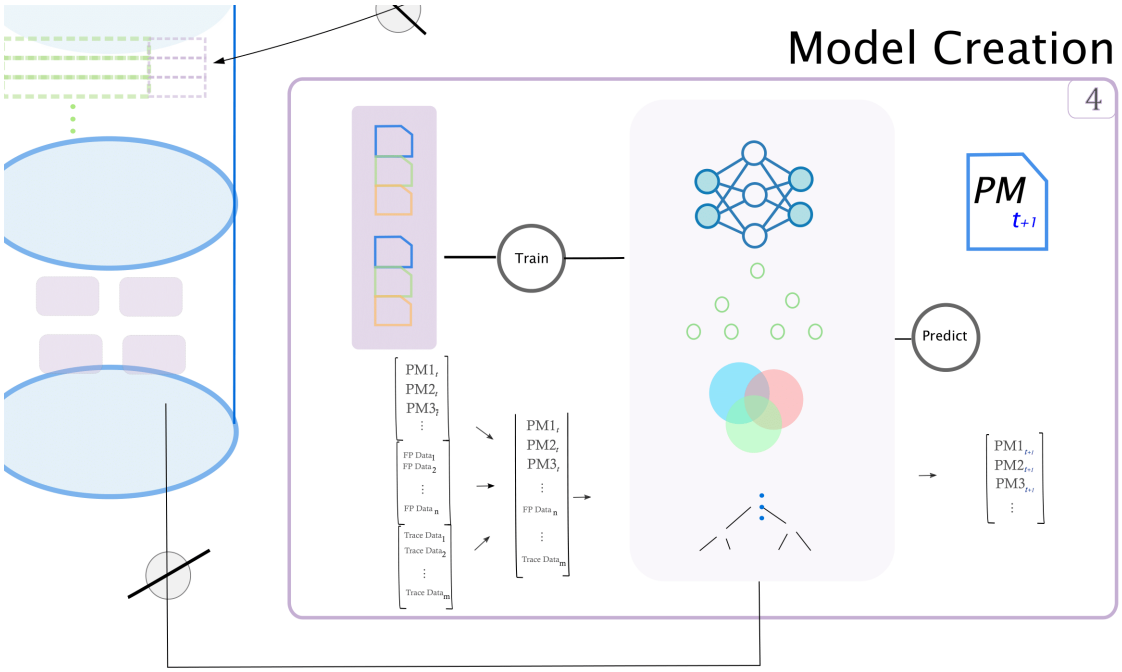

Figure 18: Fourth Step, Continuous Model Creation and Evaluation.

The model creation part, also takes part in a notebook, where the results of the model can be easily accessible, the process is also designed to be able to train a model, and save it until new samples are collected, in this way also promoting a continuous model training, thus also a continuous model creation process.

4.5 Feature Selection

The feature selection is one of the key processes to discover predictive power on the model creation process — three feature selection methods were implemented on the model creation pipeline of this thesis.

- the Filtering method using Pearson's correlation criteria to eliminate the features that didn't provide much information, see Section 3.3.1.1
- the Embedded method using importance weight from Random Forest Regressors (RFR), see Section 3.3.1.3
- and the Embedded method seen as an optimization problem using linear model LASSO. see Section 3.3.1.3

From the three methods finally the last one was the only one which its feature selection eventually lead up to a model that had considerable predictive power, this being an empirical research, will try further to discuss and hypothesize why LASSO performed better than the other in the task of feature selection, with this particular kind of data, see Section 6.1.

After the feature selection step from the pre-processing options combinations, only four were able to continue toward the model creation step:

- $P + E$, (*did not pass feature selection*)
- $P + E2$, (*did not pass feature selection*)
- $P + F$, (*did not pass feature selection*)
- $P + F2$, (*did not pass feature selection*)
- $P + F3$, (**good results**)
- $P + F + E$, (*did not pass feature selection*)
- $P + F2 + E$, (*did not pass feature selection*)
- $P + F3 + E$, (**good results**)
- $P + F2 + E2$, (*don't have good results*)
- $P + F3 + E2$, (**good results**)

4.6 Algorithms

The python library scikit-learn provides a handful of machine learning algorithms intending to achieve a complete data science project, including feature selection, model selection, data preprocessing, and many algorithms for regression (Pedregosa et al., 2011). The algorithms used in this thesis work will be presented in this section, with a brief code snippet displaying the parameter used for the model training. These parameters, even though are selected by an exhaustive hyper parametric search method, the set used by this method was discovered empirically until the best set of parameters produced stable results. After this process, a reduced set from parameters options were chosen for each algorithm.

This is the current outcome from **Cycle 5** (2.3) from this DSR process. It presents in both test configurations, the results of the generated predictive models using the different tested algorithms comparing their predictive power, measured as a higher r^2 score. The choice of these algorithms was made empirically, and they were selected as a sample of the different techniques and methods that can be relevant to construct a predictive model in the LTE L2 software system. Also, they are meant to compose a comparison basis for future implementation of other relevant algorithms as research keeps on advancing. The set of possible parameters for these algorithms was also chosen empirically and then iteratively in the style of DSR, and it was evaluated based on their r^2 scores.

4.6.1 Multi-layer Perceptron

The multi-layer perceptron algorithm, Section 3.3.4, was implemented using exhaustive Hyper parameter search, in order to look for the best combinations of parameters, the most important parameters identified in this thesis work are, the maximum amount of iterations, the activation function on the neurons, i.e., relu, identity, etc. The number of hidden layers and the learning rate.

Listing 1: Multi-layer perceptron with hyper parameters search

```
from sklearn.neural_network import MLPRegressor

param = {'max_iter':[160],
              'activation':['relu','identity'],
              'learning_rate':['adaptive'],
              'n_iter_no_change':[100],
              'hidden_layer_sizes':[(1000,)]}

model = MLPRegressor()

grid_search = GridSearchCV(model, param_grid=param, cv=7, iid=False)

start = time()
rs = grid_search.fit(in_train, out_train)
mlp_prediction = rs.predict(in_test)
print("GridSearchCV took %.2f seconds for %d candidates"
     " parameter settings." % ((time() - start)))
report(grid_search.cv_results_)
print(rs.score(in_test,out_test))
```

As we can see the Code 1 the use of the Grid Search with Cross Validation method, to select the best combination of parameters.

4.6.2 K-neighbors

The idea behind the nearest neighbor methods is to get a predefined amount of training samples nearest in distance to the new data point and predict the label of these. The number of samples is a user-defined constant like k-nearest neighbor learning rate. The distance can be measured by any metric: standard Euclidean distance is the most common choice. Neighbors-based methods are recognized as non-generalizing machine learning methods since they “remember” all of its training data.

Listing 2: Kneighbors with hyper parameters search

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
from time import time

param = {'n_neighbors':[20,18] ,'weights':['uniform','distance'],
              "algorithm":['ball_tree', 'kd_tree', 'brute'],'p':[4]}

N_neighbors = 5
accuracies = []
model = KNeighborsRegressor(n_neighbors=N_neighbors, weights='distance')

grid_search = GridSearchCV(model, param_grid=param, cv=7, iid=False)

start = time()
rs = grid_search.fit(in_train, out_train)
kneighbors_prediction = rs.predict(in_test)
report(grid_search.cv_results_)
```

As we can see the code Extract 2, the most influential parameter found are the number of neighbors, the weights, and the style of algorithm, taking into account that the number of samples is not very high, brute force is an option.

4.6.3 Random Forest

In the random forest algorithm, all trees in the ensemble are built from a sample drawn with replacement from the training set. Also, when splitting a node through the creation of the tree, the split that is chosen is not the best split anymore. Instead, the split that is selected is the best split among a random subset of all the features. The result of this randomness increases the bias of the forest, but due to the averaging, the variance decreases, compensating for the increased bias and generating a better model.

Listing 3: randomForest with hyper parameters search

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_predict, GridSearchCV

param = {'random_state':[5,10,3],"max_depth":[40,50,60],'n_estimators': [700,500]}

model = RandomForestRegressor()

grid_search = GridSearchCV(model, param_grid=param, cv=7, iid=False)

rs = grid_search.fit(in_train, out_train)
rfr_prediction = rs.predict(in_test)
report(grid_search.cv_results_)
print(rs.score(in_test,out_test))
```

As we can see the code Extract 3 the most critical parameter found empirically are the random state, the max depth of the trees, and the number of estimators.

4.6.4 Gradient Boosting

Gradient Boosting constructs an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage, a regression tree is fit on the negative gradient of the given loss function.

Listing 4: gradientBoosting with hyper parameters search

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV

param = {'loss':['ls'], 'learning_rate':[0.1,0.2,0.15], 'n_estimators':[100],
              'alpha':[0.9,0.85],'max_depth':[3,5]}

model = GradientBoostingRegressor(n_estimators=100)

grid_search = GridSearchCV(model, param_grid=param, cv=7, iid=False)

rs = grid_search.fit(in_train, out_train)
gbr_prediction = rs.predict(in_test)
report(grid_search.cv_results_)
print(rs.score(in_test,out_test))
```

As we can see the code Extract 4, the most effective loss function found is 'ls', meaning least squares, the learning rate in a range from 0.1 to 0.2, and the max depth from the regression trees.

4.6.5 Support Vector Machines

Support Vector Machines can be extended to solve classification and regression problems; the later is called Support Vector Regression (SVR). The model produced depends on a subset of the training data because the cost function of the model does not take into account any training data near to the model prediction.

Three implementation of SVR, provided by *scikit-learn* library, was explored in the model creation step:

The Epsilon Support Vector Regression, who has C (penalty) and epsilon as free parameters

Listing 5: svr with hyper parameters search

```
from sklearn import svm
from sklearn.model_selection import GridSearchCV

param = {'cache_size':[1000],
        'degree':[40],
        'kernel':['rbf','linear','sigmoid'],
        'tol':[1e-3],
        'C':[4.0,5.0,4.5],
        'epsilon':[0.1,0.09,0.11]
       }

model=svm.SVR(cache_size=1000, degree=40, kernel='rbf',tol=1e-3, C=5.0 ,epsilon=0.1)
n_iter_search = 400
grid_search = GridSearchCV(model, param_grid=param, cv=10, iid=False,scoring='r2')

rs = grid_search.fit(in_train, out_train)
SVR_pred = rs.predict(in_test)
report(grid_search.cv_results_)
print(rs.score(in_test, out_test))
```

As we can see the code Extract 5, the exhaustive hyper-parametric search is focused on the choice of the kernel, the C penalty parameter in the range of 4.0 to 5.0, and the epsilon parameter, which specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value (Pedregosa et al., 2011).

The Nu Support Vector Regression uses a parameter nu to control the number of support vectors. Nu replaces the parameter epsilon of epsilon-SVR.

Listing 6: nuSVR with hyper parameters search

```
from sklearn.model_selection import GridSearchCV
from sklearn import svm
from sklearn.svm import NuSVR

param2 = {'gamma':['scale'],
        'degree':[20],
        'kernel':['rbf','sigmoid'],
        'tol':[1e-5,1e-3,1e-6,1e-7],
        'nu':[0.8,0.9,0.95],
        'C':[0.4,0.5,0.6,0.7]
       }

regr = NuSVR(gamma='scale', C=0.4, nu=0.9, cache_size=500)
grid_search = GridSearchCV(regr, param_grid=param2, cv=5, iid=False,scoring='r2')
rs = grid_search.fit(in_train, out_train)
nuSVR_pred = rs.predict(in_test)
report(grid_search.cv_results_)
print(rs.score(in_test, out_test))
```

As we can see the code Extract 6, the set of parameters to look for the most optimal combination consists of, two kernel choices, the Tolerance for Stopping Criterion (tol), the C penalty, in the range from 0.4 to 0.7, and the nu parameter, that sets an upper and lower bound on the fraction of training errors and the fraction of support vectors respectively (Pedregosa et al., 2011).

The Linear Support Vector Regression who has flexibility in the choice of penalties and loss functions.

Listing 7: linealSVR with hyper parameters search

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVR

param = {
        'tol':[1e-4,1e-5,1e-3,1e-6,1e-7],
        'dual':[False,True],
        'loss':['squared_epsilon_insensitive'] ,
        'C':[1.0,2.0,3.0,0.8],
        'epsilon':[0.1,0.09,0.06,0.08,0.07,0.05,0.04,0.03]
       }

regr = LinearSVR(random_state=0, tol=1e-5)

grid_search = GridSearchCV(regr, param_grid=param, cv=5, iid=False,scoring='r2')
rs = grid_search.fit(in_train, out_train)
linealSVR_pred = rs.predict(in_test)
report(grid_search.cv_results_)
print(rs.score(in_test, out_test))
```

As we can see the code Extract 7, the set of parameters that are believed to have a good combination of them regarding the performance of the model are; the tol, the choice of solving the dual or primal optimization problem, the C penalty, and the epsilon in the loss function.

4.6.6 Lasso, LassoLars and ElasticNet

The LASSO is a linear model that predicts sparse coefficients. It is helpful in some circumstances due to its inclination to favor solutions with less non-zero coefficients, reducing the number of features upon which the provided solution is dependent, see Section 3.3.1.3. It has proved to be highly effective in this thesis work, especially using Cross Validation (CV).

For high-dimensional datasets including several collinear features, LassoCV is most frequently superior. However, LassoLarsCV has the advantage of exploring important values of the alpha parameter, and if the number of samples is tiny compared to the number of features, it is often faster than LassoCV (Pedregosa et al., 2011).

The Lasso method with Cross validation.

Listing 8: Lasso Cross Validation with hyper parameters search

```
from sklearn.linear_model import LassoCV
from sklearn.model_selection import GridSearchCV
from time import time
ls = list(range(3,15))

param = {"cv": ls, "max_iter": [3000],
             'fit_intercept':[True,False],
             'selection':['random','cyclic'] }
```

```
model=LassoCV()

grid_search = GridSearchCV(model, param_grid=param, cv=5, iid=False)
start = time()
rs = grid_search.fit(in_train, out_train)
lasso_pred = rs.predict(in_test)
report(grid_search.cv_results_)
print(rs.score(in_test,out_test))
```

As we can see the code Extract 8, the most changing parameter is the number of Cross validations to try with the model, they are in a range of $(3, 15)$.

Lasso using Least-angle Regression (LARS) algorithm, LARS is specially made for high dimensional data, similar to forward stepwise regression. At each step, it finds the feature most correlated with the target. When multiple features have equal correlation, instead of continuing along with the same feature, it proceeds in a direction equiangular between the features (Pedregosa et al., 2011).

Listing 9: Lasso Lars Cross Validation with hyper parameters search

```
from sklearn.linear_model import LassoLarsCV
from sklearn.model_selection import GridSearchCV
from time import time

ls = list(range(3,15))

param = {"cv": ls, "max_iter": [1000]}

model=LassoLarsCV()

grid_search = GridSearchCV(model, param_grid=param, cv=5, iid=False)

rs = grid_search.fit(in_train, out_train)
lassoLars_pred = rs.predict(in_test)
report(grid_search.cv_results_)
print(rs.score(in_test,out_test))
```

As we can see the code Extract 9, mostly the default parameters are chosen except for fixed max iteration number at 1000, and the number of cross validations to try in the model. from 3 to 15.

ElasticNet is a linear regression model trained with both ℓ_1 and ℓ_2 norm regularization of the coefficients, see Section 3.3.1.3. This combination allows for learning a sparse model where few of the weights are non-zero like Lasso, while still maintaining the regularization properties of Ridge. Like Lasso and LassoLars, ElasticNet is done with CV, to perform iterative fitting along a regularization path (Pedregosa et al., 2011).

Listing 10: Elastic Net Cross Validation with hyper parameters search

```
from sklearn.linear_model import ElasticNetCV
from sklearn.model_selection import GridSearchCV
from time import time

ls = list(range(3,15))

param = {'l1_ratio':[.02,.05,.01],"cv": ls,
              "max_iter": [3000],'fit_intercept':[True],
              'selection':['random','cyclic'] }

model=ElasticNetCV()

grid_search = GridSearchCV(model, param_grid=param, cv=5, iid=False)

rs = grid_search.fit(in_train, out_train)
elasticNet_pred = rs.predict(in_test)
```

```
report(grid_search.cv_results_)
print(rs.score(in_test,out_test))
```

As we can see the code Extract 10, on top of the CV parameters being tested by the numbers from 3 to 15, the ℓ_1 ratio, who scales and balances the penalty between ℓ_2 and ℓ_1. this parameter is in the range of 0.01 to 0.05 means that the penalty is mostly the ℓ_2.

5 Results

This chapter contains various results from the different iterations and preprocessing choices. It first shows the Exploratory Data Analysis (EDA) from the first data preprocessing option for the thesis, this done in the first cycles of the research and also due that the EDA for other iterations is visually impossible taking into account the number of features can be more than 2000. Both test configurations analysis are shown, analysis the evolution of the features through revision time.

The results of the model creation process are presented for the two test configurations, and for each of them the four models that were created up to the moment of the writing of this thesis, it is meant to show the iterative evolution of the DSR evaluation process, as every new model outperforms the others.

5.1 Exploratory Analysis Test A

It is very useful to start looking at the correlations between our three different datasets, the PM dataset, the function-level dataset, and the event-level dataset.

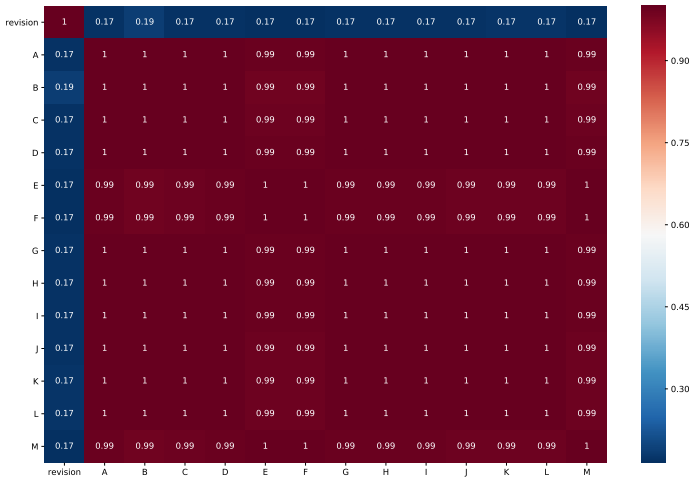

Figure 19: Correlation heat-map from PM dataset test configuration A

5.1.1 Performance measurements first part

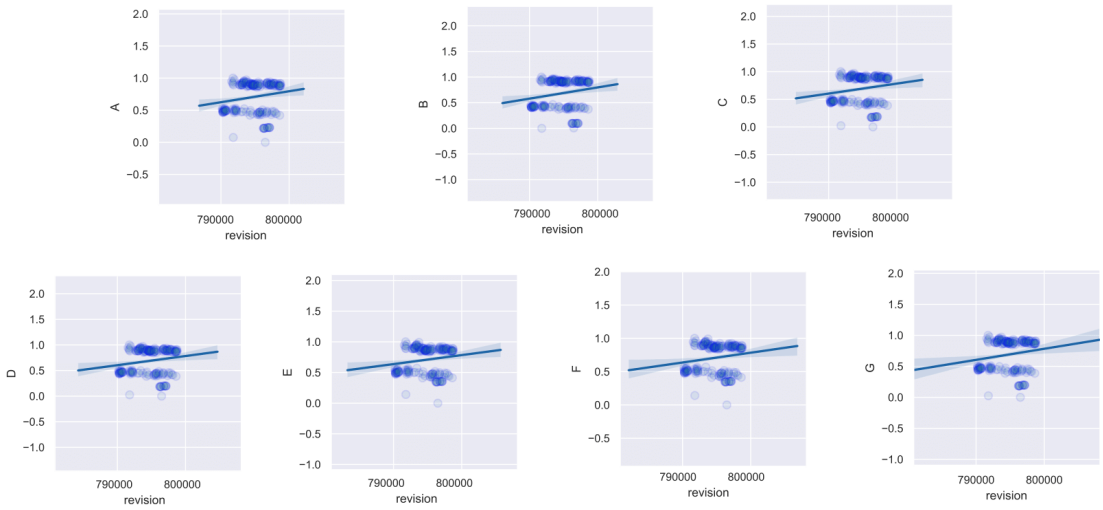

Figure 20: First seven PM evolution through revisions in test A

The correlation heat-map in Figure 19 shows that all the thirteen PM are highly correlated, suggesting they might be measuring the same or very similar properties from the L2 system, it also suggest that the predictive model has only to predict one output variable.

Taking a deeper look at the data points of the first six PM from test configuration A, Figure 20, in relation to it's evolution through time, i.e their change in different revisions, it is clear that all are sightly increasing thought time, but more interestingly it look as they measure the same property of the system, looking at the complete scatter matrix in Annexes .1, one can see how the regression line tend to have a perfect slope of 1.

The idea behind this visualization is that it allows to look for clusters of points, on top of the correlation values, which can be done by a heat-map. looking at the positions of the samples as dots enables us to distinguish where code modifications made a significant change in the performance measurement. If we observe with detail, the organization of the dots, for this test configuration, they have primarily 2 clusters, samples in the first cluster are very close to the maximum value through all revisions (from 0.8 to 1.0). And samples in the second cluster then to be close below the average value (from 0.3 to 0.5). One can also see there are two points who are very close to 0 in all the six PM values.

The simple regression line in this exploratory analysis, can already give some predictive power, and as trend shows that, there performance values are sightly increasing through time, which coincides with other observations from the system already known in Nokia.

5.1.2 Performance measurements second part

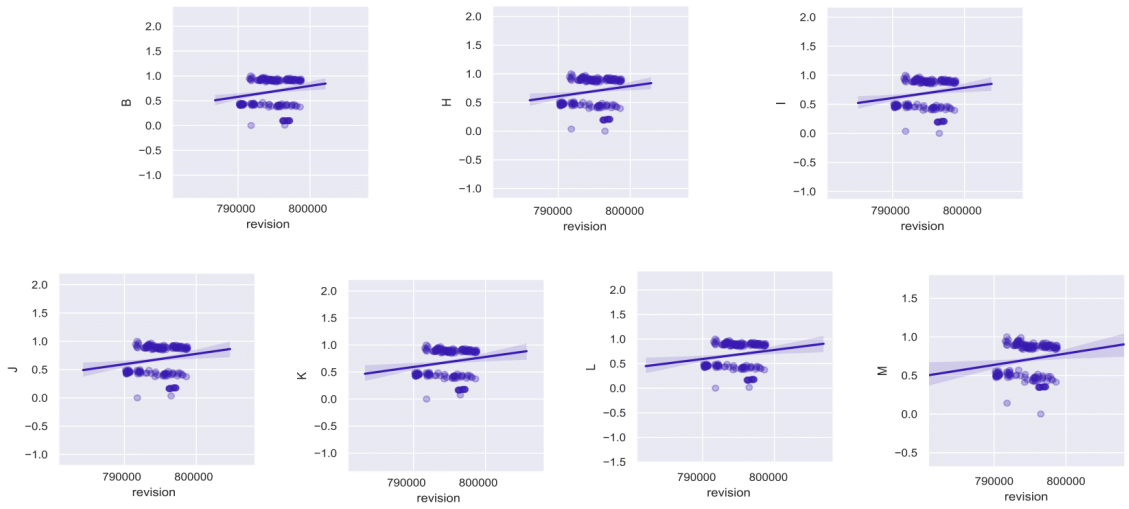

Figure 21: Last six PM evolution through revisions in test A

Continuing looking deeper at the remaining six PM, the observations are similar as the first six, all are sightly increasing thought time and its seem they are measuring the same property of the system, looking the complete scatter matrix in Annexes 48. the slopes of the regression lines tend to be 1.

Another remark from the behavior of the PM in test configuration A is that they have a threshold behavior after the mean value through all the samples, in other words the samples are never found in the range of 0.5 to 0.8, this means that the system makes the behavior jump directly to greater than 0.8 when the samples value approaches 0.5, and the same is through in the other direction.

Even though their graphs look very similar, E, F and M are more similar between them selves, and this can also be seen by the heat-map matrix in Figure 19.

5.1.3 Function-level data

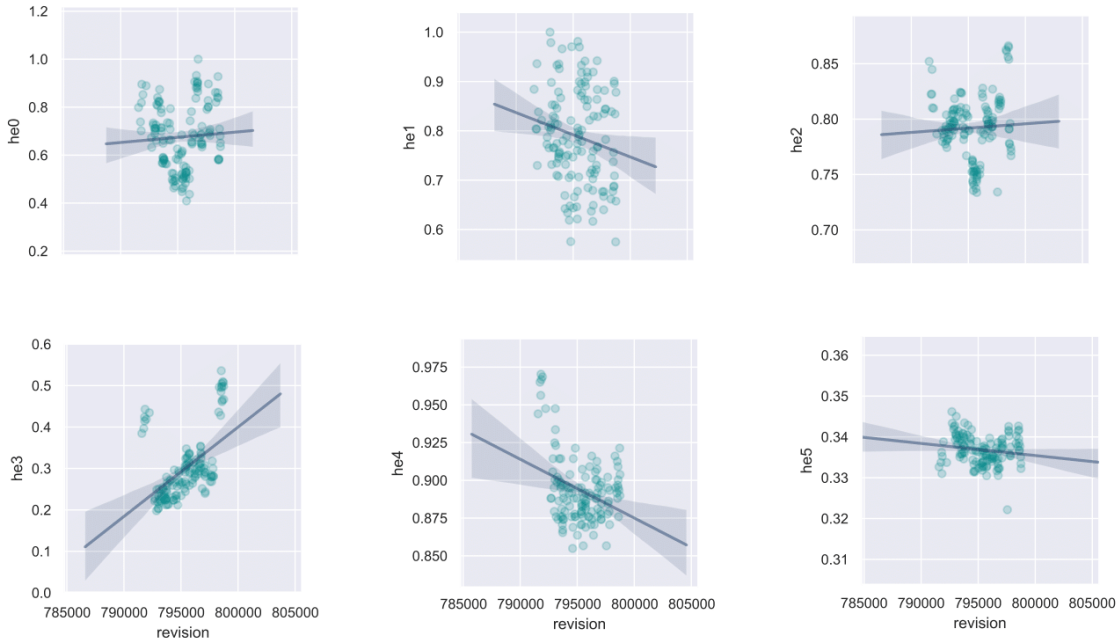

Figure 22: Function-level dataset evolution through revisions in test A.

From test configuration A, function profiling data, is clear that different hardware events are being trade off for others, the $he0$, the top scatter plot, is slightly growing thought revision evolution time, one can observe some weak clustering formations, as the transparency of the dots let us see where they are repeated most often. the graph shows how the values of the $he0$ were high at the first revision, then the decreased and when back to being high by the last revision sampled in the data.

The $he1$ hardware event by the regression line it indicates is decreasing trough time, and there is almost no clustering formation, the points look evenly distributed. this can be also appreciated in the diagonal from the scatter plot matrix found in the Annexes .2.

The $he2$ shows clearly at least three clusters, similar to $he0$, but in this case more pronounced and delineated. The $he3$ is increasing rapidly and has mostly one predominant cluster, also most of the values are below 0.5. The $he4$ and $he5$ are both is decreasing, but the former has a more variance in its values, and the later has a compact form and less decrease by its slope.

5.1.4 Event-level data

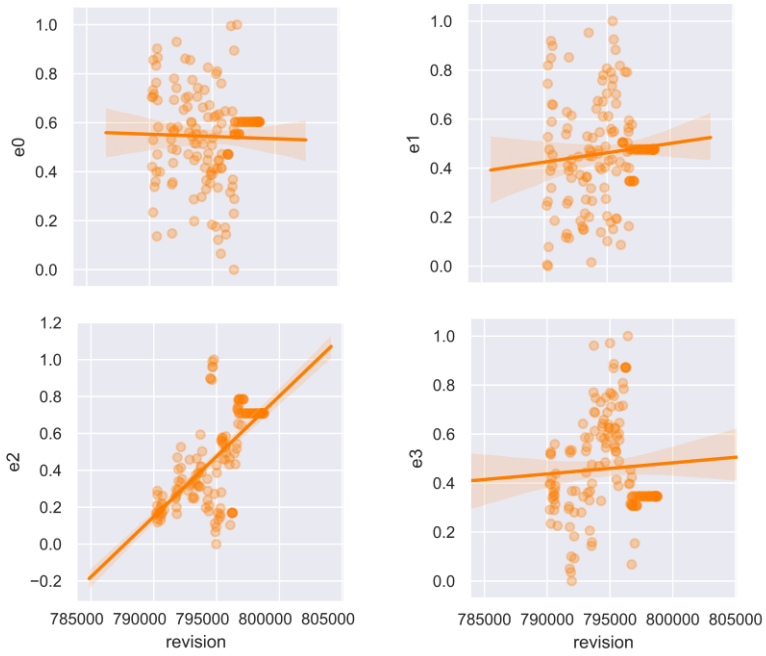

Figure 23: Event-level dataset evolution through revisions in test A.

Continuing the analysis of the results from test configuration A, the event-level data collected by tracing. The $e1$ and $e3$ are both slowly increasing through-time. $e1$ shows looks more like a normal distribution with the exception of a cluster formation by the end where its value tends to be constant, $e3$ is less normally distributed and has two centers of values in 0.4 and 0.6, this can be appreciated at the diagonal from the complete scatter plot matrix in Annexes 50.

The event $e2$ is increasing and has two centers, in early revision at 0.4 and in the newer revisions at 0.8. The event $e0$ is the only one decreasing in time compared to its past. A common pattern in all the four events is that the latests revisions maintain a constant value in all the times from each of the four events, this behavior is not seen on the past, this indicates a remarkable change in the system and how this events are measured.

5.2 Exploratory Analysis Test B

In this section the similar analysis is done to test configuration B dataset.

Looking at the heat-map matrix from the PM in Figure 24, all the down-link throughput variables are highly correlated with the exception of B, it is still positively correlated but only by 0.4 to most of the other variables. Another special variable is C who is correlated to the others by 0.75 but by it's correlated by 0.9 to B.

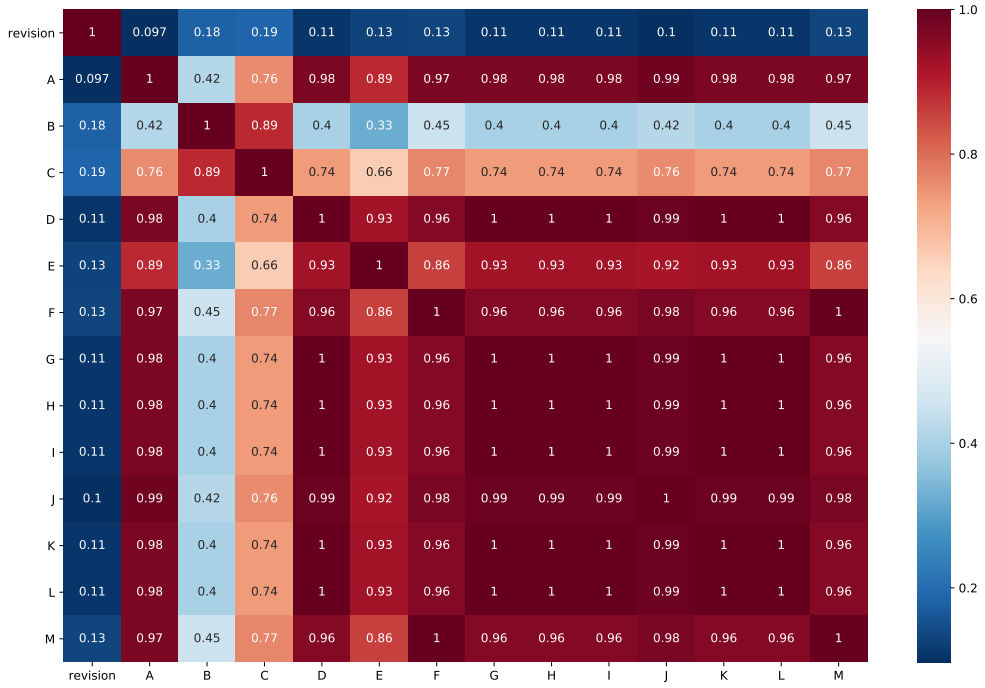

Figure 24: Correlation heat-map from PM dataset test configuration B

5.2.1 Performance measurements first part

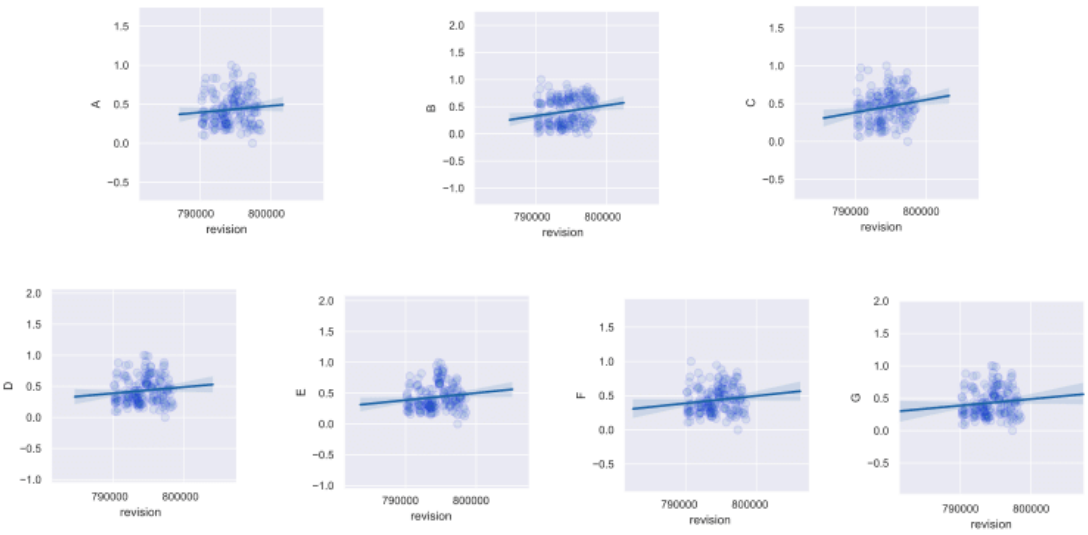

Figure 25: First seven PM evolution through revisions in test B.

Taking a deeper look at the data points of the first seven PM from test configuration A in relation to it's evolution through time, i.e their change in different revisions, it is clear that all are sightly increasing thought time, but more interestingly it looks again also in test configuration B, as they are measuring a similar property of the system, looking at the complete scatter matrix in Annexes .4 The PM B seem to be the only one who behaves differently similar to having a threshold value when after the other PM values arrive to a certain high point, B overflows and starts counting again, a very particular behavior, hard to comprehend without look into the properties each of these PM are actually measuring in the system.

By the distribution of samples in the graph one can observe the shapes they form are very similar, with the exception of B, which has could be understood as representing two normal distribution, once centered in 0.25 and the other one in 0.7, this can be appreciated by the diagonal of the matrix found in the Annexes .4.

5.2.2 Performance measurements second part

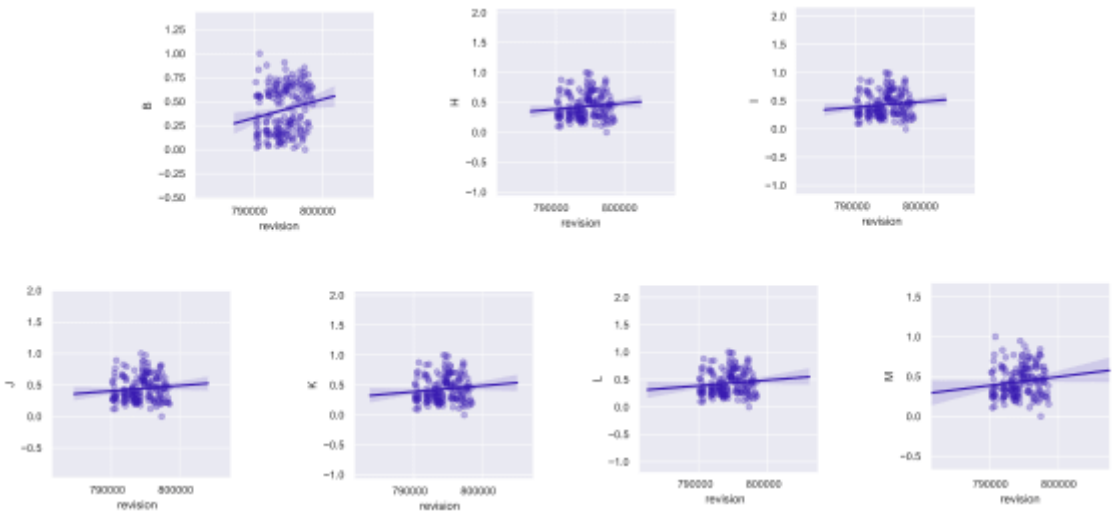

Figure 26: Last six PM evolution through revisions in test B.

From test configuration B, performance measurement data results from the last six PM plus B,it is clear again that all are sightly increasing thought time, their behavior is very similar to the first seven seen before, looking the complete scatter matrix diagonal in Annexes 52 The B seem to be the only one who behaves differently, and one can see how it reflects two normal distribution, compared mostly only one for the other six variables.

Even though their graphs look very similar, D, G, H, I, K and L graphs are more similar between them selves, and this can also be seen by the heat-map matrix in Figure 24.

5.2.3 Function-level data

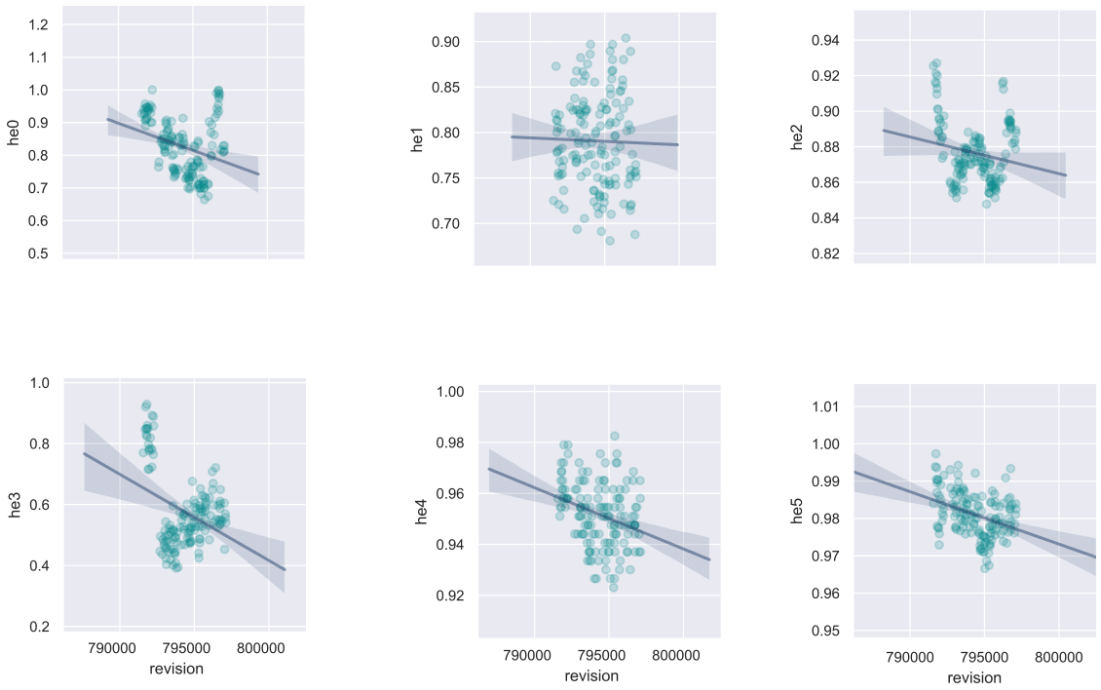

Figure 27: Function-level dataset evolution through revisions in test B.

From test configuration B, function-level profiling data, is not clear that different hardware events are being trade off for others, as its the case for test configuration A. The results show that in this test configuration all the hardware event are decreasing in different proportions. The $he0$, in the top of Figure 27, is showing clustering formations, and overall its performance value came from high to low, and then started increasing again. The graph shows how the values of the $he1$ are very disperse and its observable how the regression line has a greater interval of variance, implying it can't explain a great deal of its variance.

The $he2$ hardware event by the regression line it indicates is decreasing trough time, there is strong formations and shapes in the graph, similart to $he0$. its values were high in early revision then went low, and then starting increasing again. the complete matrix found in Annexes .5 shows how most of the six hardware event are positively correlated with the exception of $he1$ with $he0, he2, he3$ and $he4$.

The $he3$ has two clusters, and its decreasing faster than the other hardware events. The $he4$ has the particularity of its values being separated by the same gap, showing that its possible values have a different precision, they can only be at particular distances from one another. the $he5$ is decreasing as well as the other hardware events.

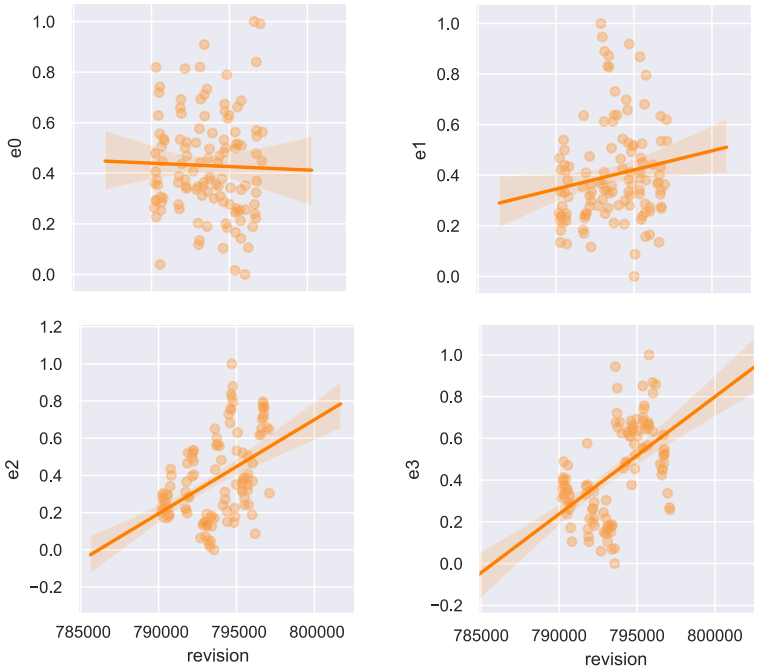

Figure 28: Event-level dataset evolution through revisions in test B.

5.2.4 Event-level data

Continuing the analysis of the data exploration from test configuration B, the event-level data collected by tracing. The $e2, e3$ and $e4$ are both increasing through-time. $e2$ and $e3$ have a similar positive slope at their regression lines, both show strong cluster formations, but in rather different values and times. taking a lot at the complete scatter plot matrix in Annexes .6, events $e0$ and $e1$ are positively correlated, equally as $e2$ and $e3$.

The event $e0$ variance is relatively constant through time, and is the only one who is not increasing, but rather slowly decreasing. There is no constant values after a specific revision as it was observed on test configuration A.

5.3 Predictive Models

There are 11 combinations from the explored preprocessing options for the function-level and event-level quantified data in regards to the DL PM. From these 11 only in 4 a model could be build passing through the feature selection process, see Figure 29 in the remaining 6 all features were removed by the feature selection process, showing already that they had no predictive power.

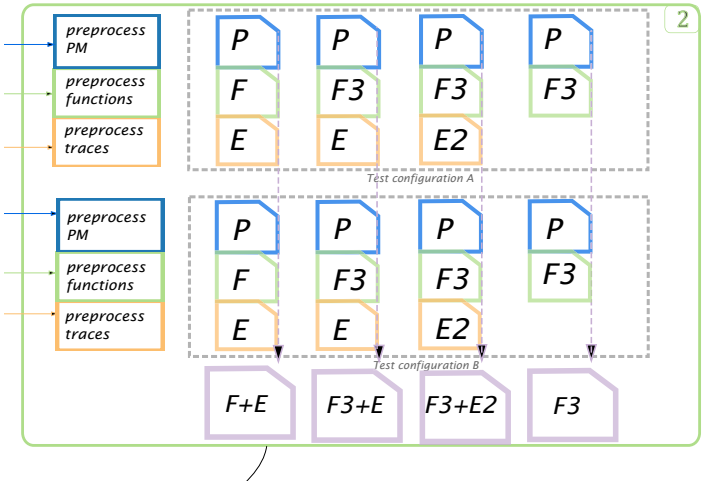

Figure 29: The four combinations presented on this thesis

The results from those four configurations are presented in this chapter, only two showed greater than 0.4 $r^2 scores$.For each viable preprocessing combination, 10 model were trained, this processed was done for both test configuration A and B, having in total an automatic creation of **80** predictive models.

Then models were trained using 75% of the dataset and they were tested using the remaining 25%, the data-set separation was done randomly, meaning the first 75% of samples is not always used for the training, but is rather different every time the processes is initiated, this to avoid, choosing by chance samples that happen to train the model better than the rest.

For each combination of data preprocessing options, two graphs and one table are presented;First A feature importance graph, displaying for the current models, the 20 most informative features, i.e the ones that provide more predictive power to the model. Second a table comparing the Coefficient of Determination (r^2) Explained Variance (EVS), Mean Squared Error (MSE), Mean Absolute Error (MAE) and the Max Error (ME) score metrics for each of the 10 algorithms, the table is sorted from the greatest r^2 score to the smallest. Finally, a third graph plotting the test values from the dependent variable, the PM A, in light blue, and joining the points by a continuous line, at the same time plotting the predicted values by six of the then algorithms[10] joint together by doted lines. This graph is relevant because it presents the results for the task of predicting values, and can provide a visual insight of what data are the model actually generating, something not possible by looking merely at metrics scores.

[10]This due to readability, the results of similar methods are just showed once, only LassoCV, no ElasticNetCV or LassoLarsCV, and it is the same case with SVR, no nuSVR or linealSVR presented in the graph, nevertheless its results can be seen in the table comparing the metrics scores

5.3.1 First Results

It is relevant to show the first result from the predictive model creation process, as we can see in Figure 30. the predictions made, by at the moment the four used algorithms, were very close to the target PM A variable, meaning the results of the r^2 scores where also very high, this was surprising at first glance, but it was due to leaving the other PM variables in the training set, thus enabling the model to have a very high predictive power, clearly because one of the variable from the PM group can be easily predicted by the others taking into account how highly correlated they are with one another. After this first experience, the other output variables where eliminated from the joint dataset, by this allowing a real exploration of the predictive potential from the others systems layers in regards the down-link throughput.

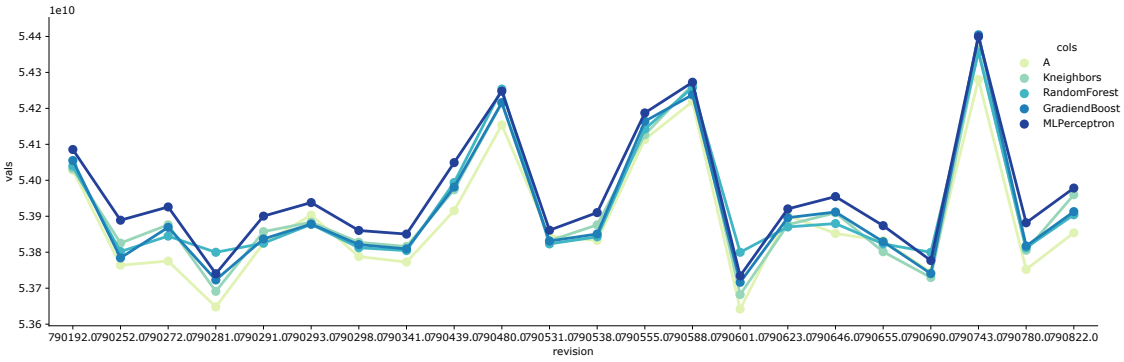

Figure 30: First iteration of the model prediction for PM test A

5.3.2 Test Configuration A

In this section results for test configuration A are presented, 40 models in total, from 4 combinations of pre-processing options.

5.3.2.1 Models $F + E$ With 164 samples, the model $F + E$ is characterized by its respective preprocessing choices, 4 event-level specially chosen variables, and 6 function-level variables, these grouped together by each HPC type of event, recorded by profiling, see Section 4.2.1.

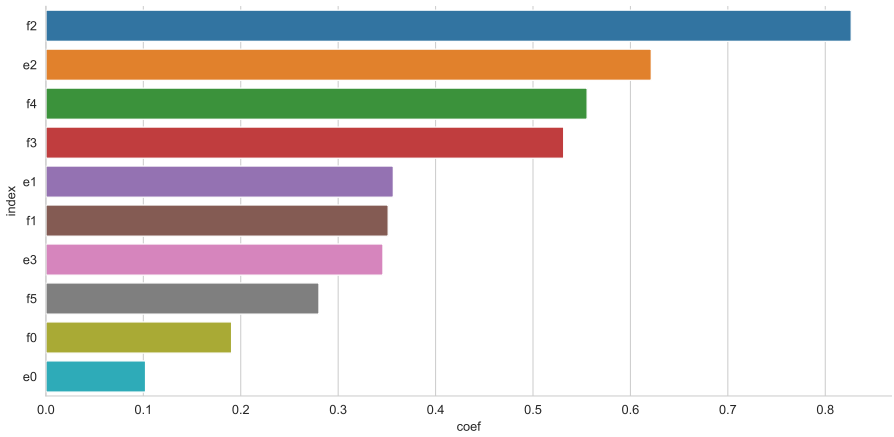

Figure 31: Feature importance ranking for $F + E$ dataset in Test A.

In Figure 31, one can see only the 10 most important features, taking into account that's the total number of input features for this model $F + E$.

The results for the 10 created models can be seen in Table 7. It is clear that the models are arbitrarily bad, the table shows LassoLarsCV having the least worst score, and nuSVR having the worst score overall, this results are only useful as an example of a combination of pre-processing options that provides no predictive power.

Table 7: Model evaluation by metrics for $F + E$ dataset in Test A.

algorithm\score	R2	EVS	MSE	MAE	ME
LassoLarsCV	-0.050288	-0.050276	0.097545	0.287545	0.682958
linealSVR	-0.070363	-0.069759	0.099409	0.293362	0.676207
RandomForest	-0.132926	-0.104515	0.105220	0.272891	0.712521
MLPerceptron	-0.133967	-0.130312	0.105316	0.303227	0.674479
ElasticNetCV	-0.155178	-0.150113	0.107286	0.306634	0.671555
GradiendBoost	-0.169759	-0.169754	0.108641	0.277329	0.706247
LassoCV	-0.170535	-0.164307	0.108713	0.309004	0.669728
Kneighbors	-0.276653	-0.276478	0.118568	0.316299	0.753851
SVR	-0.280083	-0.276832	0.118887	0.309330	0.753588
nuSVR	-0.565828	-0.564863	0.145425	0.336177	0.796853

The Figure 32 shows us the values predicted by the models, and it is visible that these predicted values have not even a tendency similar to the target output variable A.

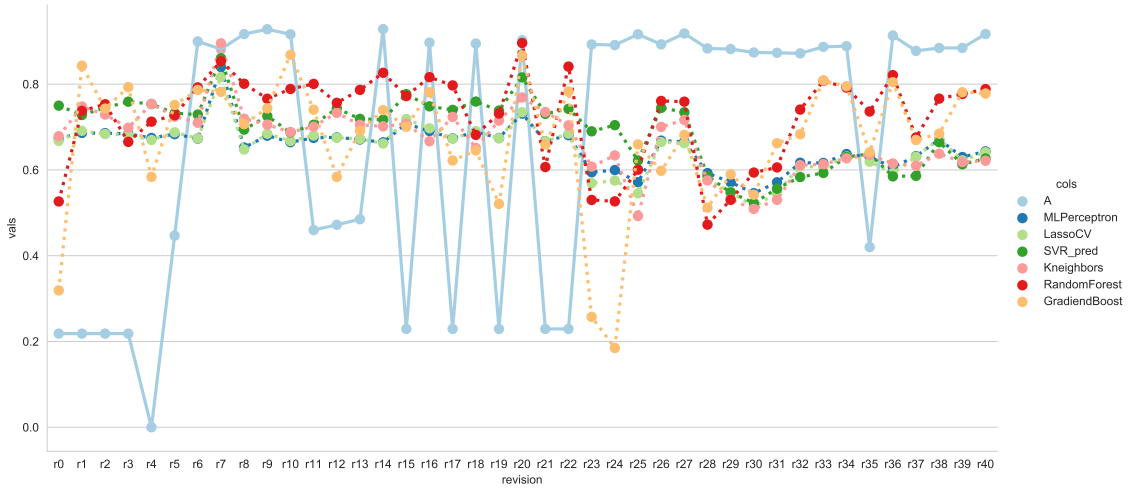

Figure 32: Prediction graph for $F + E$ dataset in Test A.

5.3.2.2 Models $F2 + E2$ With 181 samples, the model $F2 + E2$ is characterized by having more features than the previous dataset, 25 function-level and 15 event-level variables respectively, see Section 4.2.1. After the feature selection process 18 variables remained in the dataset, see Figure 33.

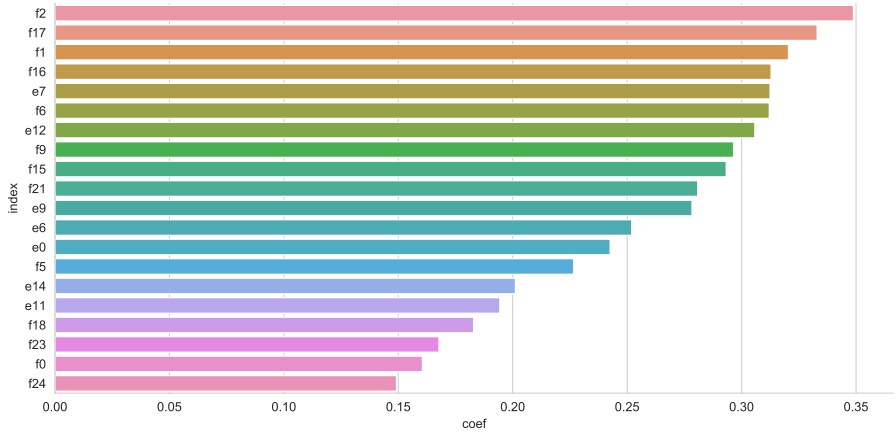

Figure 33: Feature importance ranking for $F2 + E2$ dataset in Test A.

The result from the 10 algorithms can be seen in Table 8, The SVR algorithm performs the best compared to the others, but the r^2 scores are still no even at 0.3 to claim a weak correlation, it is important to note that the results are still better than the first models with $F + E$.

The Figure 34, shows the slow evolution of the models predictive power in regards the target output variable, as it can be seen the predicted values are far from the real values, but they are better compared to models generated by $F + E$.

Table 8: Model evaluation by metrics for $F2 + E2$ dataset in Test A.

algorithm\score	R2	EVS	MSE	MAE	ME
SVR	0.119278	0.124809	0.052892	0.194819	0.591568
MLPerceptron	0.071312	0.082474	0.055772	0.208509	0.577491
ElasticNetCV	0.067469	0.103687	0.056003	0.221200	0.492699
LassoLarsCV	0.051531	0.079722	0.056960	0.219532	0.518429
nuSVR	0.051515	0.052527	0.056961	0.198893	0.625885
LassoCV	0.046352	0.083672	0.057271	0.222166	0.496566
linealSVR	0.033483	0.067527	0.058044	0.222019	0.522300
Kneighbors	0.004601	0.054448	0.059778	0.226316	0.513256
RandomForest	-0.042536	-0.009378	0.062609	0.228524	0.492562
GradiendBoost	-0.227512	-0.082850	0.073718	0.243455	0.512528

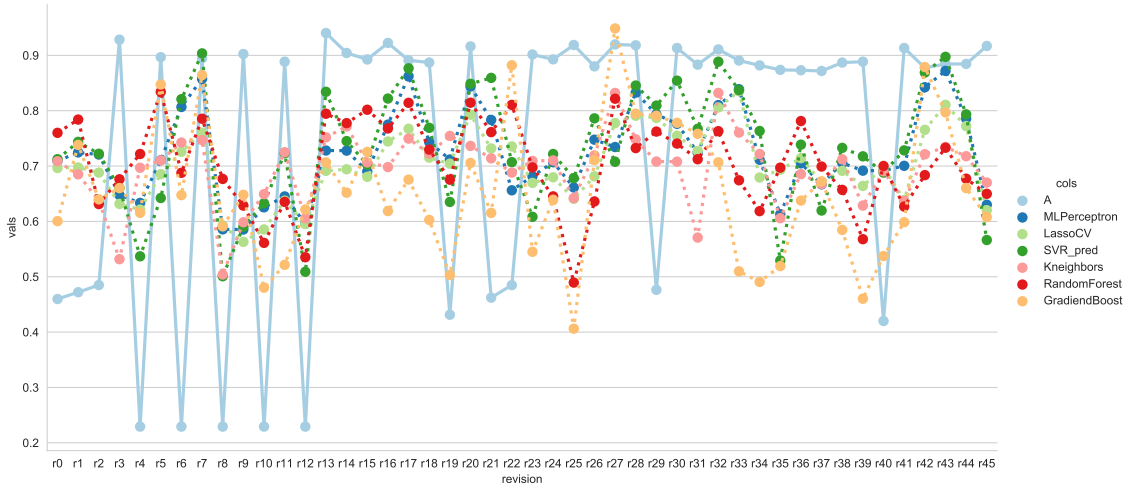

Figure 34: Prediction graph for $F2 + E2$ dataset in Test A.

5.3.2.3 Models $F3 + E2$ With 147 samples, the model $F3 + E2$ has the most high dimensional dataset from all the combinations of pre-processing options, it has 3844 function-level and 15 event-level variables respectively, see Section 4.2.1. After the feature selection process 28 variables remained in the data-set, see Figure 35.

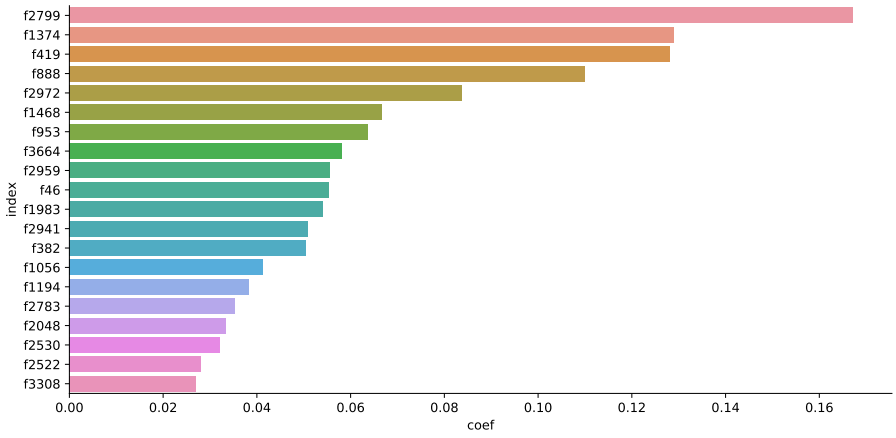

Figure 35: Feature importance ranking for $F3 + E2$ dataset in Test A.

The results from this models, Table 9, are the first results that pass the mark of 0.4 at the r^2 scores, the SVR algorithm having the best r^2 score with 0.45 and a EVS of 0.53.

Table 9: Model evaluation by metrics for $F3 + E2$ dataset in Test A.

algorithm\score	R2	EVS	MSE	MAE	ME
SVR	0.451518	0.537036	0.045970	0.187749	0.472182
linealSVR	0.438116	0.510825	0.047093	0.190477	0.493658
MLPerceptron	0.400454	0.468057	0.050250	0.194917	0.541518
ElasticNetCV	0.317200	0.466752	0.057227	0.212921	0.519018
nuSVR	0.236894	0.398550	0.063958	0.214805	0.536903
LassoCV	0.201500	0.402095	0.066925	0.226555	0.594409
LassoLarsCV	-0.129905	0.244770	0.094701	0.265511	0.693255
GradiendBoost	-0.303812	0.336924	0.109276	0.255601	0.683560
Kneighbors	-0.351354	0.237821	0.113261	0.280853	0.630779
RandomForest	-0.427148	0.232879	0.119613	0.276182	0.688922

The predicted values from the MLPerceptron, LinearSVR and SVR show a closer resemblance to the real values from the output variable as seeing in Figure 36.

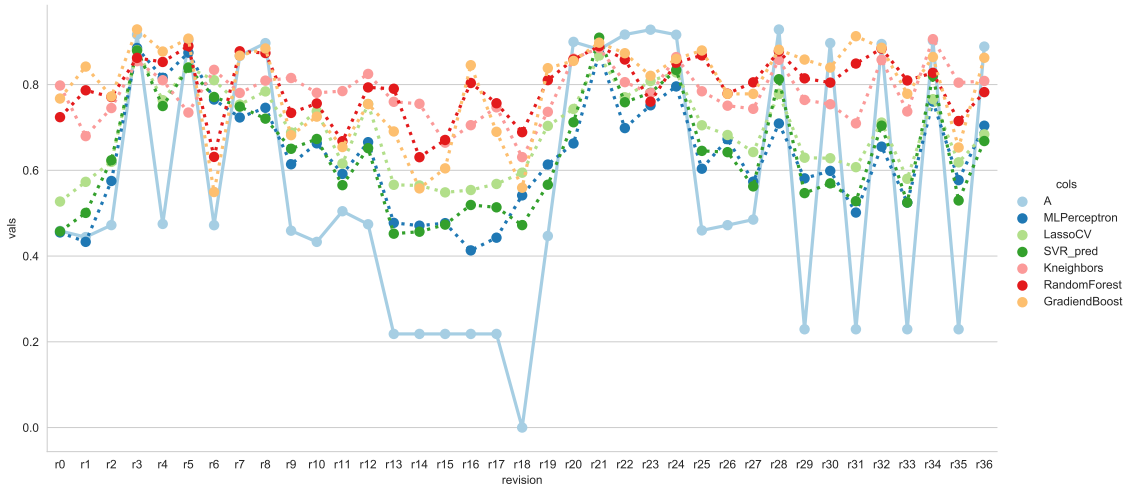

Figure 36: Prediction graph for $F3 + E2$ dataset in Test A.

5.3.2.4 **Models** $F3$ With 155 samples, the model $F3$ model is the same dataset as the previous model without the event-level data, this allowing 8 more samples in the dataset, this due to the joint process from the different layer data-sets, the fact of not having the event-level data, allowed to consider samples that where not found in the tracing dataset, see Section 4.2.1. After the feature selection process 39 variables remained in the dataset, see Figure 37

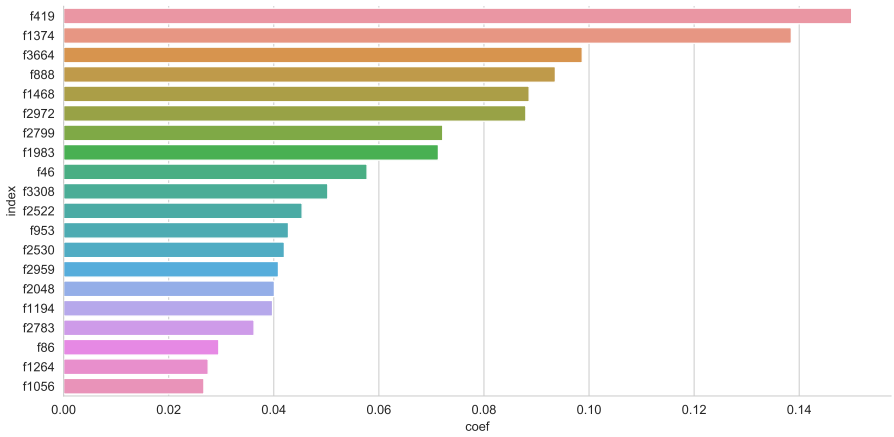

Figure 37: Feature importance ranking for $F3$ dataset in Test A.

The results from these models are the best from all other results presented in this thesis, they can be found in Table 10. All algorithms had a performance greater than 0.4 with the exception of 0.37 by RandomForestRegressor. The LassoLars with CV showed a r^2 score of 0.67, this means it can explain up to 67% of the down-link throughput variables, in this case represented by the PM A.

Table 10: Model evaluation by metrics for $F3$ dataset in Test A.

algorithm\score	R2	EVS	MSE	MAE	ME
LassoLarsCV	0.674512	0.693834	0.021510	0.126017	0.282649
LassoCV	0.670392	0.690731	0.021782	0.127372	0.291076
ElasticNetCV	0.659905	0.674282	0.022475	0.127292	0.328937
linealSVR	0.655410	0.675260	0.022773	0.133383	0.284140
SVR	0.649235	0.658767	0.023181	0.125371	0.324472
nuSVR	0.634508	0.648006	0.024154	0.120253	0.383465
MLPerceptron	0.584733	0.600312	0.027443	0.144896	0.341411
Kneighbors	0.558267	0.559404	0.029192	0.133536	0.461449
GradiendBoost	0.432479	0.450120	0.037505	0.149261	0.393780
RandomForest	0.379491	0.396336	0.041007	0.169142	0.458324

In the final Figure 38 it is very clear how all models follow the form from the output variable, finally showing the final state of evolution from the continuous model creation improvement.

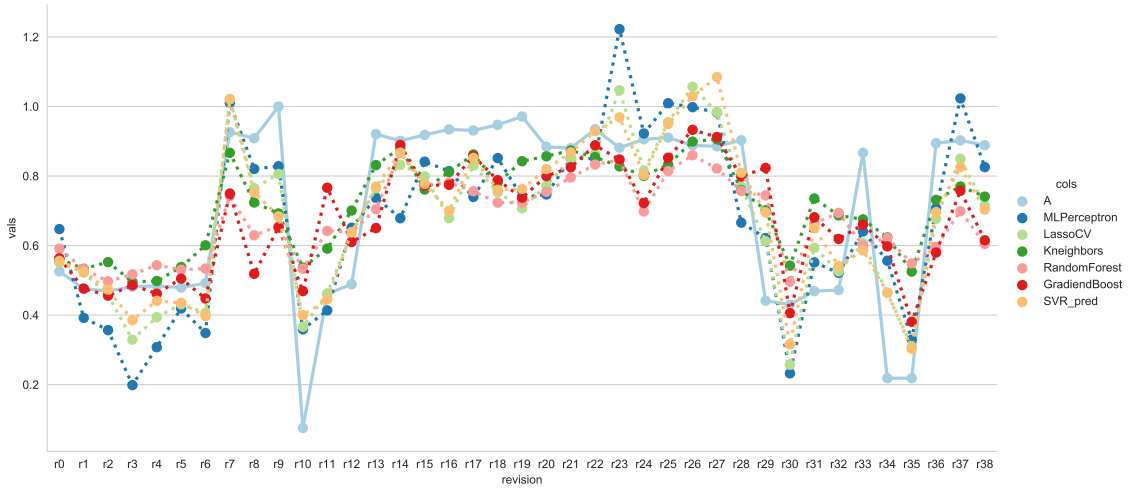

Figure 38: Prediction graph for $F3$ dataset in Test A.

5.3.3 Test Configuration B

In this section results for test configuration B are presented, 40 models in total, from 4 combinations of pre-processing options.

5.3.3.1 Models $F + E$ With 126 samples, the model $F + E$ is characterized by its respective preprocessing choices, 4 event-level specially chosen variables, and 6 function-level variables, these grouped together by each HPC type of event, recorded by profiling, see Section 4.2.1.

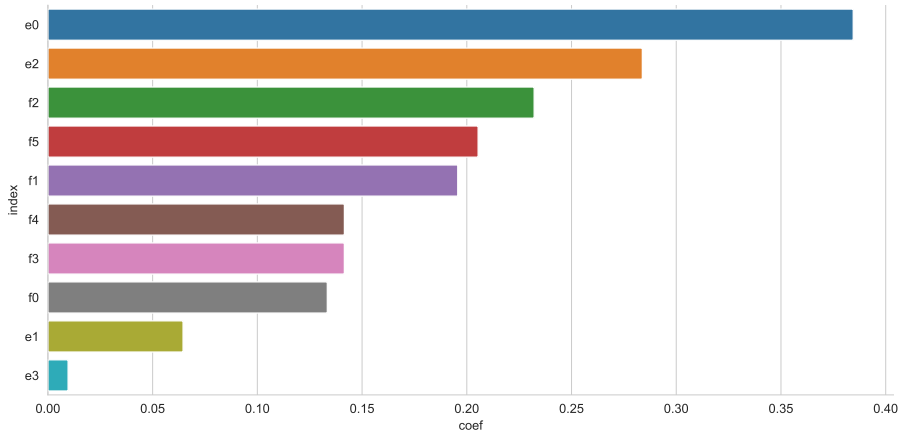

Figure 39: Feature importance ranking for $F + E$ dataset in Test B.

In Figure 39, one can see only the 10 most important features, taking into account that's the total number of input features for this model $F + E$.

The results for the 10 created models can be seen in Table 11. It is clear that the models are arbitrarily bad, the table shows LassoLarsCV having the least worst score, and GradientBoosting has the worst score overall, this results, similar as with test configuration A,

are only useful as an example of a combination of pre-processing options that provides no predictive power.

Table 11: Model evaluation by metrics for $F + E$ dataset in Test B.

algorithm\score	R2	EVS	MSE	MAE	ME
LassoLarsCV	-0.000138	0.000000	0.038278	0.167782	0.364293
ElasticNetCV	-0.000138	0.000000	0.038278	0.167782	0.364293
LassoCV	-0.000138	0.000000	0.038278	0.167782	0.364293
SVR	-0.052613	-0.049162	0.040286	0.169637	0.420303
linealSVR	-0.094629	-0.067299	0.041894	0.171447	0.383784
Kneighbors	-0.110703	-0.109941	0.042509	0.181838	0.370947
MLPerceptron	-0.126511	-0.102670	0.043114	0.173749	0.403157
nuSVR	-0.275005	-0.266339	0.048797	0.179123	0.674126
RandomForest	-0.538349	-0.481676	0.058876	0.196433	0.494961
GradiendBoost	-0.706466	-0.678994	0.065310	0.203941	0.538791

The Figure 40 shows us the values predicted by the models, and it is visible that these predicted values have not even a tendency similar to the target output variable A. This is useful to see the evolution of the model creation process.

40

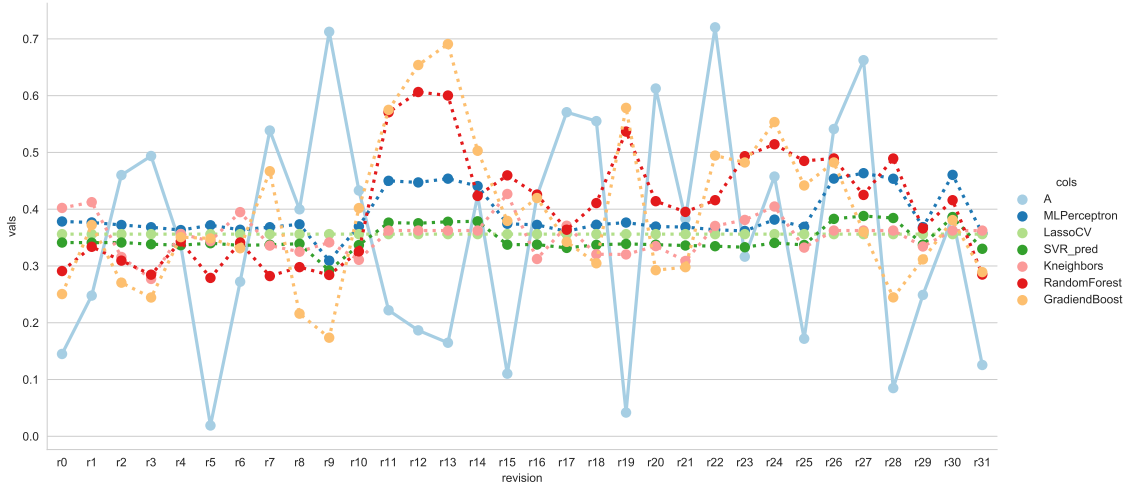

Figure 40: Prediction graph for $F + E$ dataset in Test B.

5.3.3.2 Models $F2 + E2$ With 184 samples, the model $F2 + E2$ is characterized by having more features than the previous dataset, 10 function-level and 15 event-level variables respectively, see Section 4.2.1. After the feature selection process 18 variables remained in the dataset, see Figure 41.

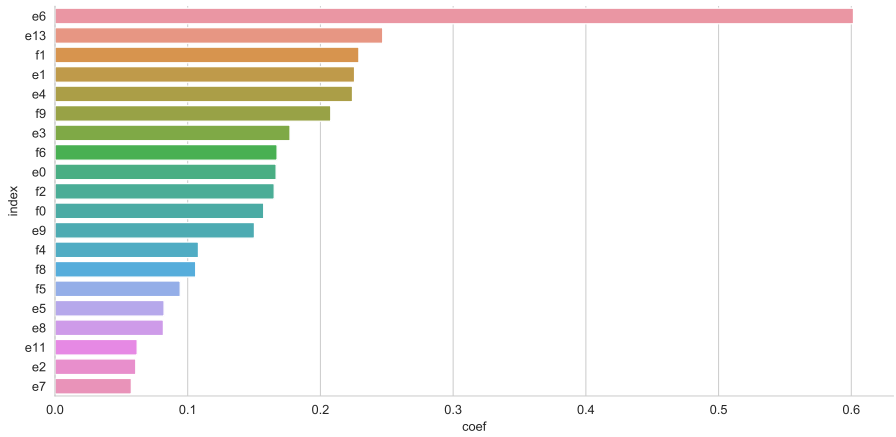

Figure 41: Feature importance ranking for $F2 + E2$ dataset in Test B.

The result from the 10 algorithms can be seen in Table 12, the results are still better than the first models with $F + E$, but they are worst than the ones from test configuration A.

Table 12: Model evaluation by metrics for $F2 + E2$ dataset in Test B.

algorithm\score	R2	EVS	MSE	MAE	ME
Kneighbors	-0.037854	-0.031083	0.041027	0.169202	0.433864
ElasticNetCV	-0.090130	-0.066591	0.043093	0.171418	0.473031
LassoLarsCV	-0.105345	-0.075251	0.043695	0.171841	0.486177
nuSVR	-0.121526	-0.121032	0.044334	0.176288	0.462097
SVR	-0.125938	-0.115034	0.044509	0.175289	0.481514
MLPerceptron	-0.136977	-0.086553	0.044945	0.172772	0.506106
LassoCV	-0.157273	-0.121906	0.045747	0.176219	0.500182
linealSVR	-0.164655	-0.103804	0.046039	0.175156	0.513932
GradiendBoost	-0.238768	-0.190531	0.048969	0.175452	0.568894
RandomForest	-0.332509	-0.293583	0.052675	0.185253	0.602771

The Figure 42, shows the evolution of the models predictive power in regards the target output variable, they are arbitrarily bad, as their r^2 scores suggest.

42

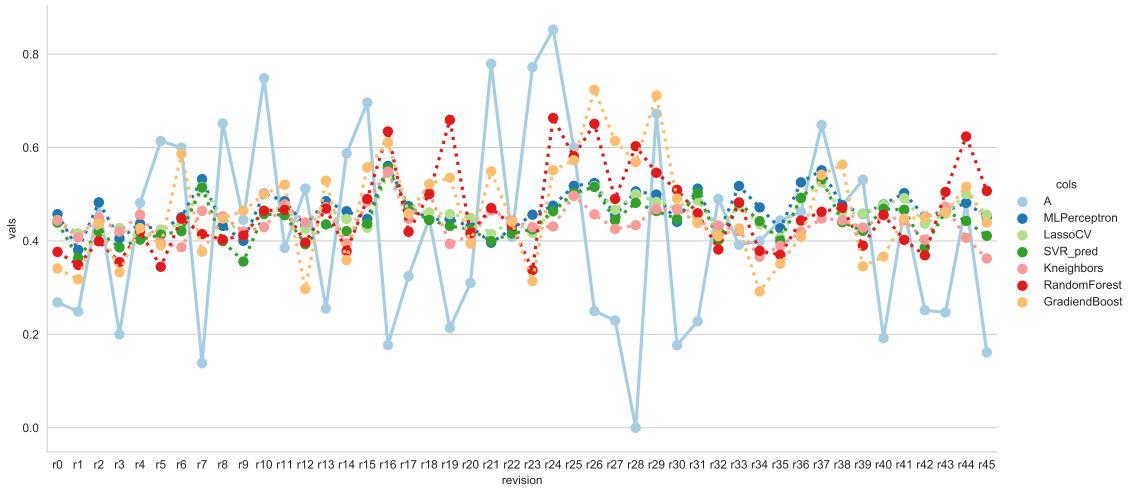

Figure 42: Prediction graph for $F2 + E2$ dataset in Test B.

5.3.3.3 Models $F3 + E2$ With 160 samples, the model $F3 + E2$ has the most high dimensional dataset from all the combinations of pre-processing options, it has 2363 function-level and 14 event-level variables respectively, see Section 4.2.1. After the feature selection process 34 variables remained in the data-set, see Figure 35.

43

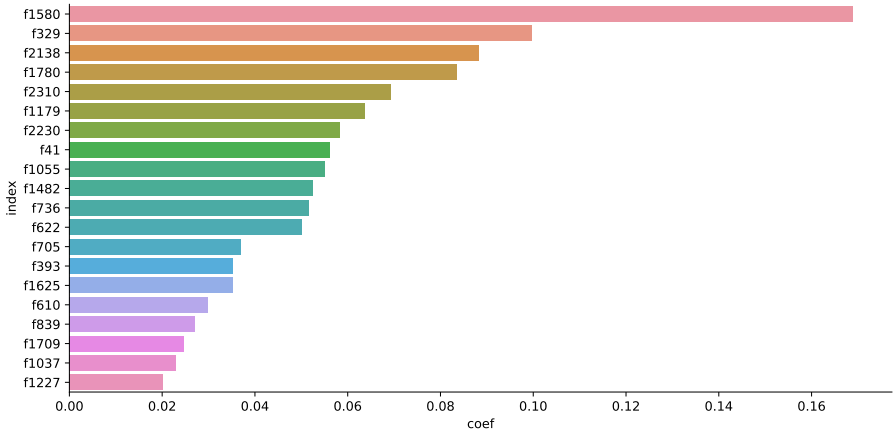

Figure 43: Feature importance ranking for $F3 + E2$ dataset in Test B.

The results from this models, Table 13, are the first results that all r^2 scores are positive and 6 pass the mark of 0.4, the Multi-layer perceptron algorithm has the best r^2 score with 0.58 and a EVS of 0.58. In comparison to test configuration A, the results with this combination of preprocessing options are better, implying results variance depending on the test scenarios where the data was recollected.

Table 13: Model evaluation by metrics for $F3 + E2$ dataset in Test B.

algorithm\score	R2	EVS	MSE	MAE	ME
MLPerceptron	0.584798	0.587412	0.016331	0.099638	0.302688
linealSVR	0.547794	0.547956	0.017787	0.104087	0.322929
SVR	0.499271	0.523452	0.019695	0.111203	0.327180
ElasticNetCV	0.459393	0.502902	0.021264	0.115091	0.354204
LassoCV	0.411017	0.448779	0.023166	0.119077	0.368606
LassoLarsCV	0.405961	0.429946	0.023365	0.116465	0.379009
nuSVR	0.328046	0.417269	0.026430	0.129131	0.377836
Kneighbors	0.317803	0.319057	0.026833	0.127591	0.446919
GradiendBoost	0.268061	0.297659	0.028789	0.125585	0.396206
RandomForest	0.126963	0.154322	0.034339	0.145061	0.422461

The predicted values from the SVR, LinearSVR and MLPerceptron, similar as in test configuration A, have the best r^2 scores and they show an even closer resemblance to the real values from the output variable, see Figure 44.

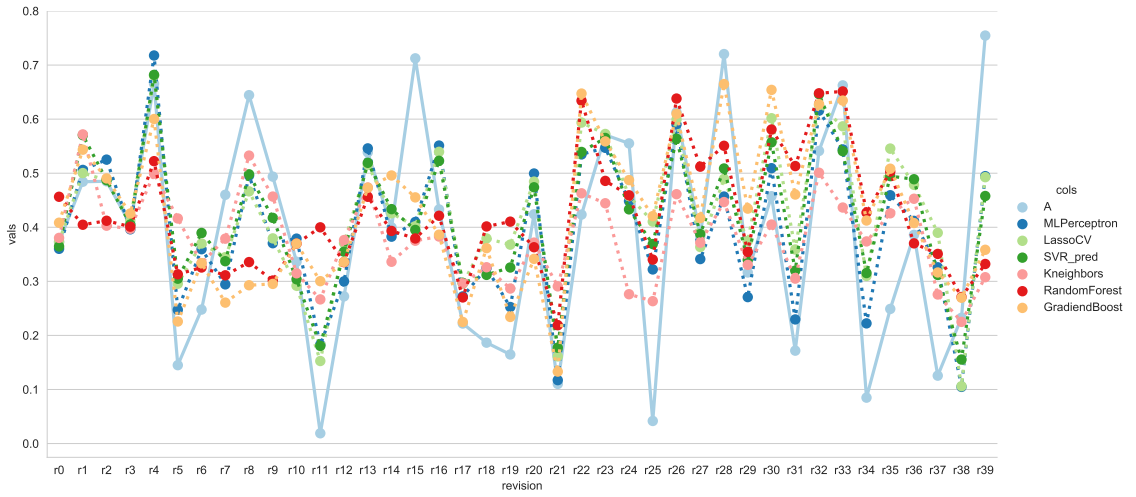

Figure 44: Prediction graph for $F3 + E2$ dataset in Test B.

5.3.3.4 Models $F3$ With 164 samples, the model $F3$ model is the same dataset as the previous model without the event-level data, this allowing 4 more samples in the dataset, this due to the joint process from the different layer data-sets, the fact of not having the event-level data, allowed to consider samples that where not found in the tracing dataset, see Section 4.2.1. After the feature selection process 36 variables remained in the data-set, see Figure 45

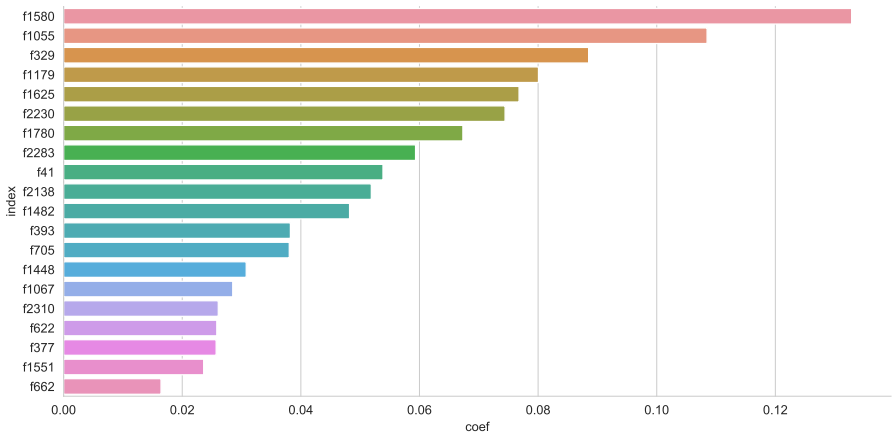

Figure 45: Feature importance ranking for $F3$ dataset in Test B.

The results from these models almost as good as the previous presented in models $F3 + E2$, they can be found in Table 14. Seven algorithms had a performance greater than 0.4. The linealSVR has a r^2 score of 0.54, this means it can explain up to 54% of the down-link throughput variables, in this case represented by the PM A.

In the Figure 46 it is clear how most models follow the form from the target output variable, showing the current state of predictive power in test configuration B.

Table 14: Model evaluation by metrics for $F3$ dataset in Test B.

algorithm\score	R2	EVS	MSE	MAE	ME
linealSVR	0.545412	0.602935	0.019907	0.109323	0.321150
ElasticNetCV	0.508825	0.552283	0.021509	0.114063	0.316024
nuSVR	0.504390	0.542753	0.021703	0.117800	0.330222
MLPerceptron	0.497121	0.537165	0.022022	0.120163	0.346045
SVR	0.492214	0.539623	0.022237	0.118698	0.293016
LassoLarsCV	0.453925	0.501895	0.023913	0.118748	0.328816
LassoCV	0.435644	0.482747	0.024714	0.121355	0.334910
Kneighbors	0.172196	0.300014	0.036251	0.157813	0.440875
RandomForest	0.125728	0.216587	0.038286	0.157815	0.504521
GradiendBoost	0.050610	0.163337	0.041575	0.164075	0.489410

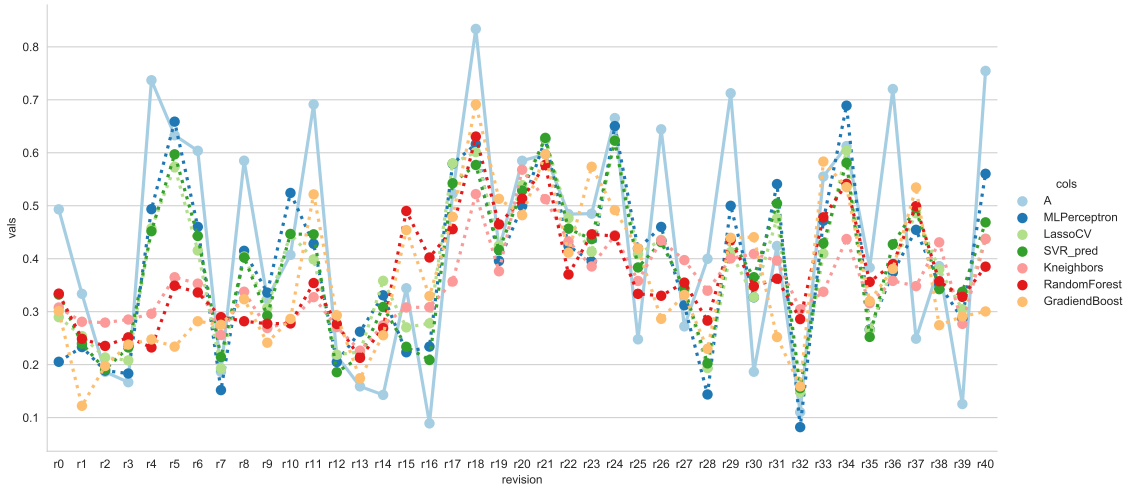

Figure 46: Prediction graph for $F3$ dataset in Test B.

5.4 Model creation Process Evaluation

The model creation process evaluation is done by comparing the results of new created models to previous ones, after this work, there exist already reference models to be compared with, in the case of a continuation of the use of this approach to increase the understanding of different layers to a global property of the LTE L2 system. This is the current outcome from **Cycle 6** (2.3) from this DSR process, by the constant evaluation the models and the architecture of the whole process.

6 Discussion

6.1 Model Creation Process

The feature selection part of the process as well as the raw data pre-processing were the crucial parts of the process that either lead up to a model with non-predictive power or one who does. It is interesting to discuss the different characteristics from the decisions on these two steps that allowed a positive outcome at the final predictive model.

6.1.1 Raw Data Preprocessing

The pre-processing options that showed better results have a much higher dimensionality than the ones who showed very weak or were not even able to show any results. This is in agreement by other research done in Nokia by Laivamaa (2019), in which the author states how despite having a dataset with a large number of features, more than 2000, the anomaly algorithm performed better with this dataset morphing rather than a most current low-dimensional dataset.

The difference between having a low dimensional dataset and a high dimensional is the system's precision level to take into account. For example, the analysis of a car by how many doors, tires and cylinders in the motor has a different utility than describing the same car very detail characteristics, for example, the material, radio, texture, weight of the tire; many details and parts that make up a functioning car motor. The LTE L2 system, at least at the function-level layer, seems to be more usefully described by high dimensional data, every function combined with the CPU core it executing it and also the special hardware event it can trigger.

6.1.2 Feature Selection

The feature selection decision for LASSO with cross-validation was driven mostly by empirical results; it was the one in which it has a feature reduction produced a model with real predictive power. Filtering methods as well as other embedded methods were tried, but their results were not successful, this does not mean they are not adapted for this kind of study, this means at the current level of experimentation and research of this thesis they showed no usefulness. This may require a deeper understanding and adaptation to be effective.

Another feature selection algorithm used was linear SVR which in theory is not conceptually very distant from the LASSO method, this method was proven to reduce the features by approximately half, and it was used when the LASSO features selection algorithm would have eliminated almost all features from the model. This is already an indicator that the specific pre-processed dataset combination cannot generate a model with good predictive power, but despite this, the model creation process was continued, and as expected those model had little or no existing predictive power.

It is interesting to state that the reduced features after a successful feature selection

process, tend not to be correlated between them, which makes sense if understood as it eliminates redundant variables that do not offer new information, this can be seen in Annexes 55 and .8.

6.2 Research Questions Answers

- Main RQ: **How can we implement a predictive model to fulfill the future needs based on the quantified data collected from different sources from the LTE L2 system?**

  The answer to the main RQ is the proposed architecture for the whole process required to create a useful predictive model, see Figure 14, in this case, to understand relations and dependencies from different system layers and the down-link throughput emergent system property. This approach can be used to analyze other emergent properties of the system as well, modifying the data recollection step to meet the new requirement needs.

- $RQ2$: **What are most essential elements affecting the down link throughput in LTE L2 system?**

  After the model creation process and looking at the ones with most predictive power, for test configuration B, 34 functions were identified out of 2363. The values from these functions can be multiplied by the values of their corresponding linear coefficients generated by a linear predictive model such as LASSO, thus being able to explain up to 54% of the variances from down-link throughput variables. For test configuration A, 28 functions were identified out of 3844. The values from these functions can also be multiplied by the values of their corresponding linear coefficients generated by a linear predictive model, thus also being able to explain up to 67% of the variances from down-link throughput variables in this test configuration. The similarity of the r^2 scores suggests that the performance could similar for other test configurations.

- $RQ3$: **How can we evaluate and present the predicted model to ensure it fulfills the future needs?**

  The evaluation of the generated predicted models is done by comparison to the past generated predictive models, having in the architecture of the whole model creation process a base method to compare new models to referent ones.

- $RQ3.1$: **How good can functions-level performance data predict and model down-link throughput related PM?**

  The function-level high-dimensional data produces models with good predictive power, in both test configuration the generated model can explain up to 58% and 67% of the variation on the down-link throughput performance measuring variables, this is on the range from moderate to strong predictive power whose r^2 scores are in the range from 0.5 to 0.7.

- $RQ3.2$: **How good can tracing signals performance data predict and model down-link throughput related PM?**

  The actual event tracing pre-processing choices alone do not produce a model with any predictive power, nevertheless, in test configuration B being joint with

the proper function-level dataset($F3 + E2$), it improved the results slightly from the predictive models, compared to the models generated by the same function-level dataset alone($F3$). This does not mean that event-level quantified data alone cannot produce a model with predictive power but rather the adequate handling of this data, if it exists, has not been found.

6.3 Validity of Results

Threats to the validity can appear from changing or unstable test results data; this means that running the same test at the same revision could provide different results. This threat is believed to be relatively low as the tests seem to be stable in the testing environment, nevertheless to deal with this threat, in the exploration analysis step the test results datasets were constructed in two manners. In one hand, it was done by averaging multiple same revision test results. And on the other hand, they were joint by the first occurrence, meaning without averaging. The results from the predictive models were unchanged.

The risk for over-fitting is always present in machine learning, that is why CV was used in feature selection and in the model creation when available, to ensure the model is not just memorizing the data. To ensure accurate results from the predictive models, the separation from training data (75%) and test data (25%) was done randomly, in other words, the training values were not always the first values by revision, with this avoiding a non representative performance of the model due to the static selection of the training data.

6.4 Future Work

After various attempts on combinations from differently pre-processed data, features coming from the function-level performance data were identified to explain the variance from the target variable that represents the DL throughput from the L2 software system. Providing a prove of concept. The complex relationships of data coming from two layers of a telecommunications system such as the SOI studied in this work can be discovered by the creation process of a predictive model, the critical aspects from this process are, the raw data pre-processing, and the feature selection.

Having this as a basis, the research of the affectation of other tests configurations to the down-link throughput can be the next step. Another next step is the further exploration of raw-data morphing or pre-processing that will enable the creation of a better predictive model, the actual results of 58% to 67% of variance explained leave room for improvements.

The exploration of the affectation of other emergent properties of the LTE L2 system, by analysis of other PM related to other properties can be beneficial to learn more about the effectiveness of this approach.

The integration of this continuous generation of predictive models can be integrated to the CI pipeline, by rapidly testing the selected functions to obtain an estimate of the affectation of the new code changes to the target system property, in the case of this

thesis, the down-link throughput.

7 Conclusions

The use of a predictive model to address the inherent complexity growth and be able to increase the predictability of a complex system's performance was explored in this work. Having positive results and much room for improvement of such results. The model creation process proposed architecture was driven multi-layer data approach. The creation of such model was possible by joining up together quantified data coming from different layers exploring the transformations of the different datasets, and an appropriate feature selection step, identifying from all the variables in the dataset, which ones can predict the down-link throughput, represented by PM associated to its performance.

The results, after various iterations and attempt with different pre-processing approaches and feature selection techniques, of the best predictive models were moderate approaching high predictive power, implying that the steps of feature selection and pre-processing are the most sensible. The exploration and use of various data visualization techniques to present the predictive power of the created models effectively was done. As well as a technical exploration of the best adapted tools to manage the automated creation of various visualizations.

The ability to obtain a quick estimate of the impact of new software functions and code modifications before its integration to a release and finished is possible by the approach proposed by this thesis. This is enabling a whole range of benefits such as a better planning of the future requirements, an accurate description of the quality attributes, and a better characterization of the system in development. A proof of concept has been presented, showing that this kind of analysis is possible and that a quantified software engineering focus is beneficial to have a better understanding of the actual state of the developed system. There is much future work to be done in order to consolidate and improve the current results of the continuously generated models.

7.1 Summary

The research for a more accurate model for a complex system such as the LTE L2 was carried out by DSR methodology, defining the requirements of the process that has an outcome a useful predictive model. First, an introduction to the problem was presented, defining its scope as the LTE L2 system, and stating the desired contribution of a deeper understanding of the affectation between different layers enabled by the generated artifact.

The research questions were stated as a guide for the DSR process. The background, from an understanding of the LTE system to different machine learning an visualization techniques, was conducted. A model creation process was proposed and constructed iteratively, consisting of four steps; Data recollection, pre-processing, exploratory analysis, and the model creation. In this process, feature selection algorithms, pre-processing data options, and different algorithms were explored and empirically selected. The results of the exploratory analysis and the predictive models were also presented iteratively, up to the best current predictive model. The answers to the research questions were stated, and positive results were found in the constructed models. On the four different pre-processed data combinations 10 algorithms were compared and the models trained with

the best combinations were able to explain the variance from the down-link throughput performance variables up to 54% and 67% respectively in the two system configurations studied, these results provide a comparative basis towards a constant improvement of the predictive model creation process.

8 References

3GPP 36.321, T. (2019, May). *Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification.* Retrieved from https://www.etsi.org/deliver/etsi_ts/136300_136399/136321/15.05.00_60/ts_136321v150500p.pdf

3GPP 36.322, T. (2018, July). *Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Link Control (RLC) protocol specification.* Retrieved from https://www.etsi.org/deliver/etsi_ts/136300_136399/136322/15.01.00_60/ts_136322v150100p.pdf

3GPP 36.323, T. (2019, May). *Evolved Universal Terrestrial Radio Access (E-UTRA); Packet Data Convergence Protocol (PDCP) Specification.* Retrieved from https://www.etsi.org/deliver/etsi_ts/136300_136399/136323/15.03.00_60/ts_136323v150300p.pdf

3GPP 36.425, T. (2019, May). *Telecommunication management; Performance Management (PM); Performance measurements Evolved Universal Terrestrial Radio Access Network (E-UTRAN).* Retrieved from https://www.etsi.org/deliver/etsi_ts/136300_136399/136323/15.03.00_60/ts_136323v150300p.pdf

Arthur, D., & Vassilvitskii, S. (2007). k-means++: The Advantages of Careful Seeding. , 11.

Asthana, A., & Olivieri, J. (2009, May). Quantifying software reliability and readiness. In *2009 IEEE International Workshop Technical Committee on Communications Quality and Reliability* (pp. 1–6). Naples, FL, USA: IEEE. Retrieved from http://ieeexplore.ieee.org/document/5137352/ doi: 10.1109/CQR.2009.5137352

Athanasiou, D., Nugroho, A., Visser, J., & Zaidman, A. (2014, November). Test Code Quality and Its Relation to Issue Handling Performance. *IEEE Transactions on Software Engineering*, *40*(11), 1100–1125. Retrieved from http://ieeexplore.ieee.org/document/6862882/ doi: 10.1109/TSE.2014.2342227

Benaissa, N., Bonvoisin, D., Feliachi, A., & Ordioni, J. (2016). The PERF Approach for Formal Verification. In T. Lecomte, R. Pinger, & A. Romanovsky (Eds.), *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification* (pp. 203–214). Cham: Springer International Publishing. Retrieved from https://link.springer.com/chapter/10.1007/978-3-319-33951-1_15

Chandrashekar, G., & Sahin, F. (2014, January). A survey on feature selection methods. *Computers & Electrical Engineering*, *40*(1), 16–28. Retrieved from https://linkinghub.elsevier.com/retrieve/pii/S0045790613003066 doi: 10.1016/j.compeleceng.2013.11.024

Chen, T.-H., Syer, M. D., Shang, W., Jiang, Z. M., Hassan, A. E., Nasser, M., & Flora, P. (2017, May). Analytics-Driven Load Testing: An Industrial Experience Report on Load Testing of Large-Scale Systems. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)* (pp. 243–252). Buenos Aires, Argentina: IEEE. Retrieved from http://ieeexplore.ieee.org/document/7965448/ doi: 10.1109/ICSE-SEIP.2017.26

Dahlman, E., Parkvall, S., & Skold, J. (2011). *4g LTE/LTE-Advanced for Mobile Broadband*. Retrieved 2019-03-30, from http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10483471 (OCLC: 838879409)

E. Rumelhart, D., E. Hinton, G., & J. Williams, R. (1986). Learning Representations by Back Propagating Errors. *Nature*, *323*, 533–536. doi: 10.1038/323533a0

Fitzgerald, B., & Stol, K.-J. (2017, January). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, *123*, 176–189. Retrieved from https://linkinghub.elsevier.com/retrieve/pii/S0164121215001430 doi: 10.1016/j.jss.2015.06.063

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal of Statistical Software*, *33*(1). Retrieved from http://www.jstatsoft.org/v33/i01/ doi: 10.18637/jss.v033.i01

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning* (1st ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. Retrieved from https://dl.acm.org/citation.cfm?id=534133&qualifier=LU1043988

Gregg, B. (2016, March). The Flame Graph. *Queue*, *14*(2), 10:91–10:110. Retrieved from http://doi.acm.org/10.1145/2927299.2927301 doi: 10.1145/2927299.2927301

Gregg, B. (2019, May). *Stack trace visualizer. Contribute to brendangregg/FlameGraph development by creating an account on GitHub.* Retrieved 2019-05-31, from https://github.com/brendangregg/FlameGraph (original-date: 2011-12-16T02:20:53Z)

Guyon, I., & Elisseeff, A. (2003, March). An Introduction to Variable and Feature Selection. *J. Mach. Learn. Res.*, *3*, 1157–1182. Retrieved from http://dl.acm.org/citation.cfm?id=944919.944968

Hevner, A., & Chatterjee, S. (2010). Design Science Research in Information Systems. In *Design Research in Information Systems* (Vol. 22, pp. 9–22). Boston, MA: Springer US. Retrieved from http://link.springer.com/10.1007/978-1-4419-5653-8_2 doi: 10.1007/978-1-4419-5653-8_2

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. , 32. Retrieved from https://wise.vub.ac.be/sites/default/files/thesis_info/design_science.pdf

Hoang Thuan, N., Drechsler, A., & Antunes, P. (2019). Construction of Design Science Research Questions. *Communications of the Association for Information Systems*, 332–363. Retrieved from https://aisel.aisnet.org/cais/vol44/iss1/20 doi: 10.17705/1CAIS.04420

Holma, H., & Toskala, A. (2009). *LTE for UMTS: OFDMA and SC-FDMA Based Radio Access*. Retrieved from https://www.wiley.com/en-us/LTE+for+UMTS%3A+OFDMA+and+SC+FDMA+Based+Radio+Access-p-9780470745472

Javidian, M. A., Jamshidi, P., & Valtorta, M. (2019, February). Transfer Learning for Performance Modeling of Configurable Systems: A Causal Analysis. *arXiv:1902.10119 [cs]*. Retrieved from http://arxiv.org/abs/1902.10119 (arXiv: 1902.10119)

Kim, S.-J., Koh, K., Lustig, M., Boyd, S., & Gorinevsky, D. (2007, December). An Interior-Point Method for Large-Scale -Regularized Least Squares. *IEEE Journal of Selected Topics in Signal Processing*, *1*(4), 606–617. Retrieved from http://ieeexplore.ieee.org/document/4407767/ doi: 10.1109/JSTSP.2007.910971

Kojouharov, S. (2019, March). *Cheat Sheets for AI, Neural Networks, Machine Learning, Deep Learning & Data Science.* Retrieved 2019-05-05, from https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-science-pdf-f22dc900d2d7

Laivamaa, T. (2019). *Evaluation of machine learning algorithms for detecting software performance anomalies* (Master's thesis, University of Oulu, Faculty of Information Technology and Electrical Engineering, Department of Information Processing Science, Information Processing Science). Retrieved from http://urn.fi/URN:NBN:fi:oulu-201903071288

Lal, T. N., Chapelle, O., Weston, J., & Elisseeff, A. (2006). Embedded Methods. In I. Guyon, M. Nikravesh, S. Gunn, & L. A. Zadeh (Eds.), *Feature Extraction* (Vol. 207, pp. 137–165). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://link.springer.com/10.1007/978-3-540-35488-8_6 doi: 10.1007/978-3-540-35488-8_6

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient BackProp. In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), *Neural Networks: Tricks of the Trade: Second Edition* (pp. 9–48). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from https://doi.org/10.1007/978-3-642-35289-8_3 doi: 10.1007/978-3-642-35289-8_3

LTE. (n.d.). Retrieved 2019-03-30, from https://www.3gpp.org/technologies/keywords-acronyms/98-lte

LTE-Advanced. (n.d.). Retrieved 2019-03-30, from https://www.3gpp.org/technologies/keywords-acronyms/97-lte-advanced

Maldonado, E. d. S., Shihab, E., & Tsantalis, N. (2017, November). Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt. *IEEE Transactions on Software Engineering*, *43*(11), 1044–1062. Retrieved from http://ieeexplore.ieee.org/document/7820211/ doi: 10.1109/TSE.2017.2654244

Mens, T. (2008). Introduction and Roadmap: History and Challenges of Software Evolution. In *Software Evolution* (pp. 1–11). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from https://doi.org/10.1007/978-3-540-76440-3_1 doi: 10.1007/978-3-540-76440-3_1

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. , 9.

Molnar, I. (2009). Performance Counters for Linux, v8. Retrieved from https://lwn.net/Articles/336542/

Motakis, A., Spyridakis, A., & Raho, D. (2013, May). Introduction on performance analysis and profiling methodologies for KVM on ARM virtualization. In T. Riesgo & M. Conti (Eds.), (p. 87640N). Grenoble, France. Retrieved from http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2018086 doi: 10.1117/12.2018086

Moutsatsos, I. K., Hossain, I., Agarinis, C., Harbinski, F., Abraham, Y., Dobler, L., … Parker, C. N. (2017). Jenkins-CI, an Open-Source Continuous Integration System, as a Scientific Data and Image-Processing Platform. *SLAS DISCOVERY: Advancing Life Sciences R&D*, *22*(3), 238–249. Retrieved from https://doi.org/10.1177/1087057116679993 doi: 10.1177/1087057116679993

Muller, G. (2007). How to Characterize SW and HW to Facilitate Predictable Design? Retrieved from https://www.gaudisite.nl/info/PerformanceEngineering.info.html

Offnfopt. (2015, May). *English: Diagram of OSI model. (This image includes several translations).* Retrieved 2019-04-10, from https://commons.wikimedia.org/wiki/File:OSI_Model_v1.svg

Olivieri, J. (2012, March). Hardware and software readiness: A systems approach. In *2012 IEEE International Systems Conference SysCon 2012* (pp. 1–6). Vancouver, BC, Canada: IEEE. Retrieved from http://ieeexplore.ieee.org/document/6189444/ doi: 10.1109/SysCon.2012.6189444

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Peffers, K., Tuunanen, T., Gengler, C., Rossi, M., Hui, W., Virtanen, V., & Bragge, J. (2006). The design science research process: A model for producing and presenting information systems research. In *Desrist international conference on design science research in information systems and technology, claremont, ca, usa, february 24-25, 2006* (pp. 83–106). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.469.2936

Piippo, P. (2018). *Automatic capacity test case result analysis from CPU loads* (Master's thesis, University of Oulu, Faculty of Information Technology and Electrical Engineering, Computer Science and Engineering). Retrieved from http://urn.fi/URN:NBN:fi:oulu-201812083252

plot.ly. (n.d.). *Scatterplot Matrix.* Retrieved 2019-05-22, from https://plot.ly/python/splom/

Rougier, N. P. (2019, May). *Adaptation of Jake VanderPlas graphic about python visualization landscape: rougier/python-visualization-landscape.* Retrieved 2019-05-22, from https://github.com/rougier/python-visualization-landscape (original-date: 2017-06-12T17:19:26Z)

Schmidt, F., Niepert, M., & Huici, F. (2018, February). Representation Learning for Resource Usage Prediction. *arXiv:1802.00673 [cs]*. Retrieved from http://arxiv.org/abs/1802.00673 (arXiv: 1802.00673)

Shang, W., Hassan, A. E., Nasser, M., & Flora, P. (2015). Automated Detection of Performance Regressions Using Regression Models on Clustered Performance Counters. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering - ICPE '15* (pp. 15–26). Austin, Texas, USA: ACM Press. Retrieved from http://dl.acm.org/citation.cfm?doid=2668930.2688052 doi: 10.1145/2668930.2688052

Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, *58*(1), 267–288. Retrieved from http://www.jstor.org/stable/2346178

Toupin, D. (2011, January). Using Tracing to Diagnose or Monitor Systems. *IEEE Software*, *28*(1), 87–91. Retrieved from https://ieeexplore.ieee.org/abstract/document/5672523?ALU=LU1043988 doi: 10.1109/MS.2011.20

TracingBook - TracingWiki. (2009, February). Retrieved 2019-03-30, from https://web.archive.org/web/20090224184220/http://ltt.polymtl.ca/tracingwiki/index.php/TracingBook

VanderPlas, J. (2017). *Python's Visualization Landscape (PyCon 2017).* Retrieved 2019-05-22, from https://speakerdeck.com/jakevdp/pythons-visualization-landscape-pycon-2017

von Luxburg, U., Belkin, M., & Bousquet, O. (2008, April). Consistency of spectral clustering. *The Annals of Statistics*, *36*(2), 555–586. Retrieved from http://projecteuclid.org/euclid.aos/1205420511 doi: 10.1214/009053607000000640

Yu, C. H. (2017, November). *Exploratory Data Analysis* (Tech. Rep.). Oxford University Press. Retrieved from http://www.oxfordbibliographies.com/display/id/obo-9780199828340-0200 (type: dataset) doi: 10.1093/obo/9780199828340-0200

Zimmermann, H. (1980, April). OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, *28*(4), 425–432. Retrieved from https://ieeexplore.ieee.org/document/1094702?ALU=LU1043988 doi: 10.1109/TCOM.1980.1094702

A Appendix

.1 Performance measurements scatter matrix, test A

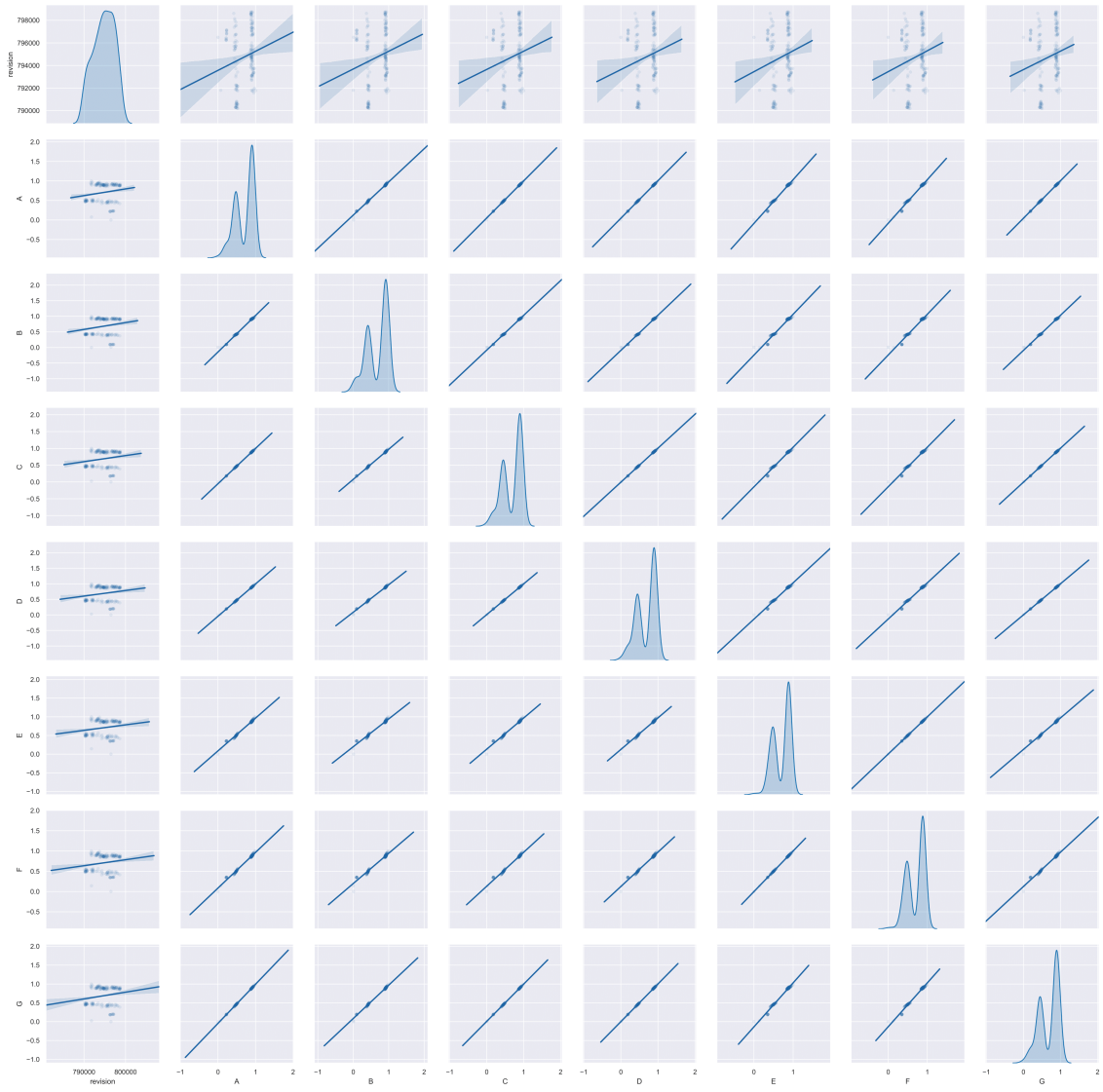

Figure 47: First seven PM scatter plot matrix with simple linear regression in test A.

The first seven PM are all highly correlated with some particular behavior in the PM B, it behaves as it has some threshold that triggers a change in its value, this could be similar to the behavior of a car gearbox

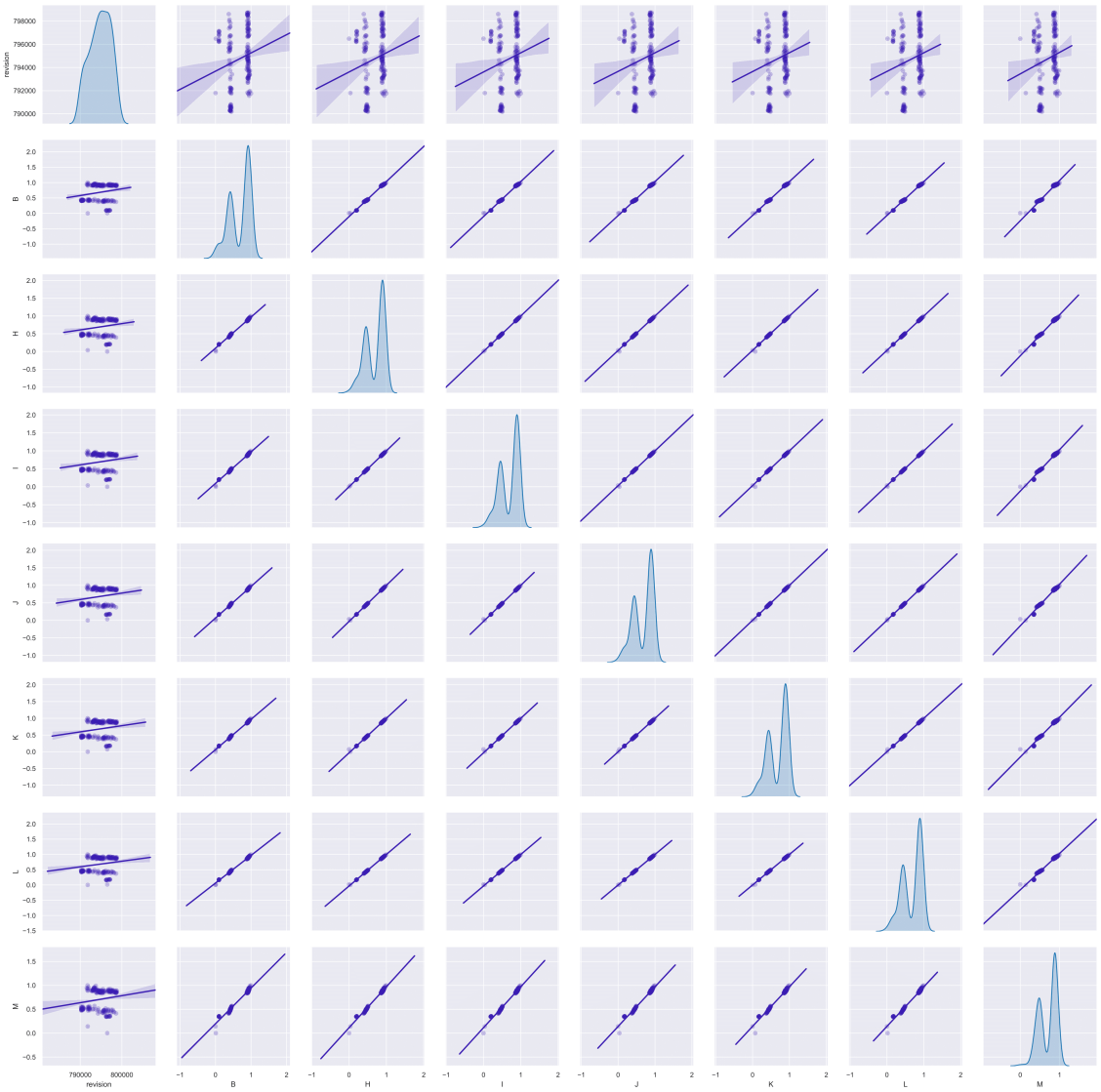

Figure 48: Last six PM scatter plot matrix with simple linear regression in test A.

Almost all performance measurements are highly correlated, one could conclude that they are actually measuring the same property of the system.

.2

Function-level scatter matrix by processor event, test A

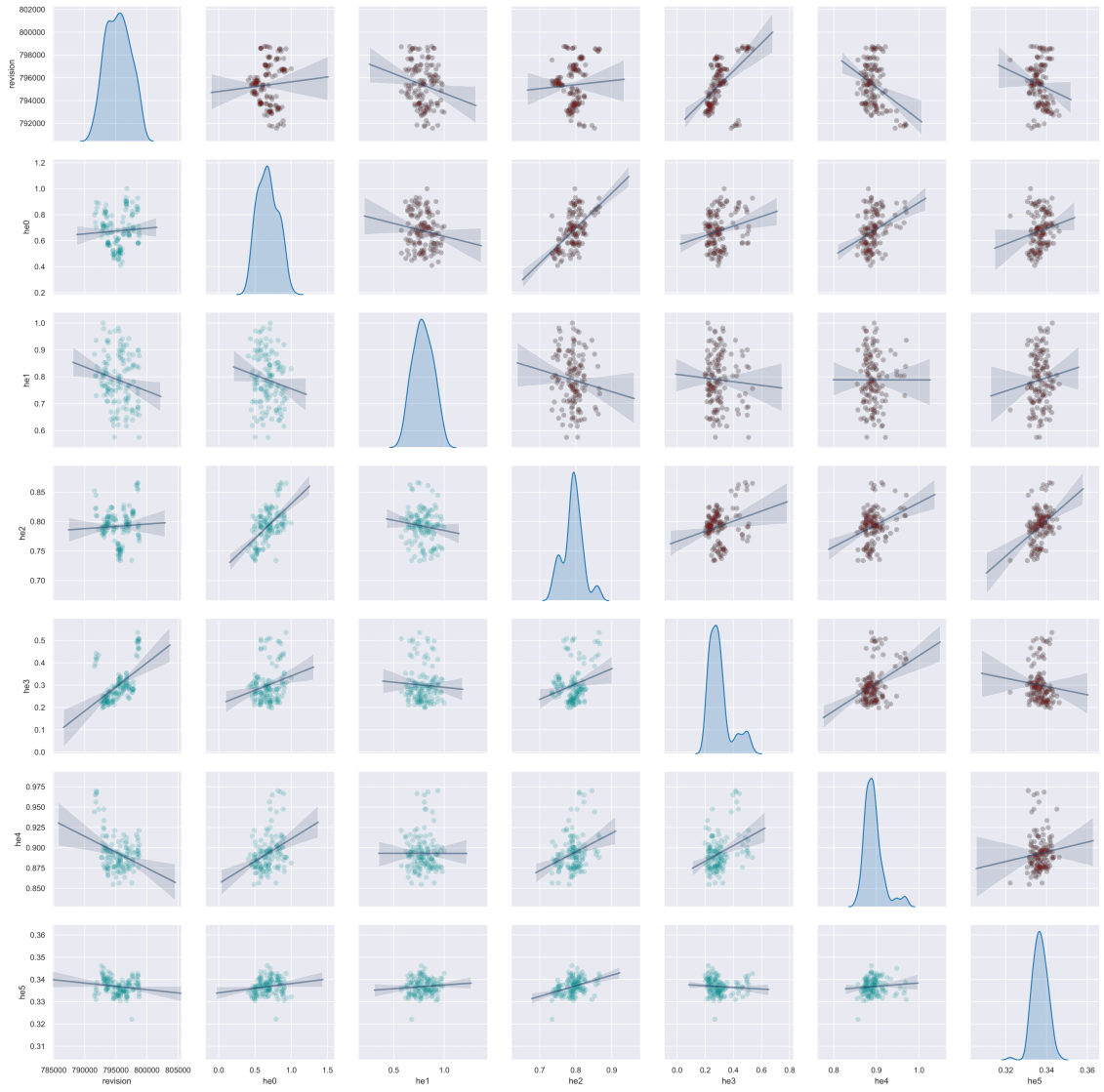

Figure 49: Function-level dataset scatter plot matrix with simple linear regression in test A.

.3

Event-level scatter matrix by selected events, test A

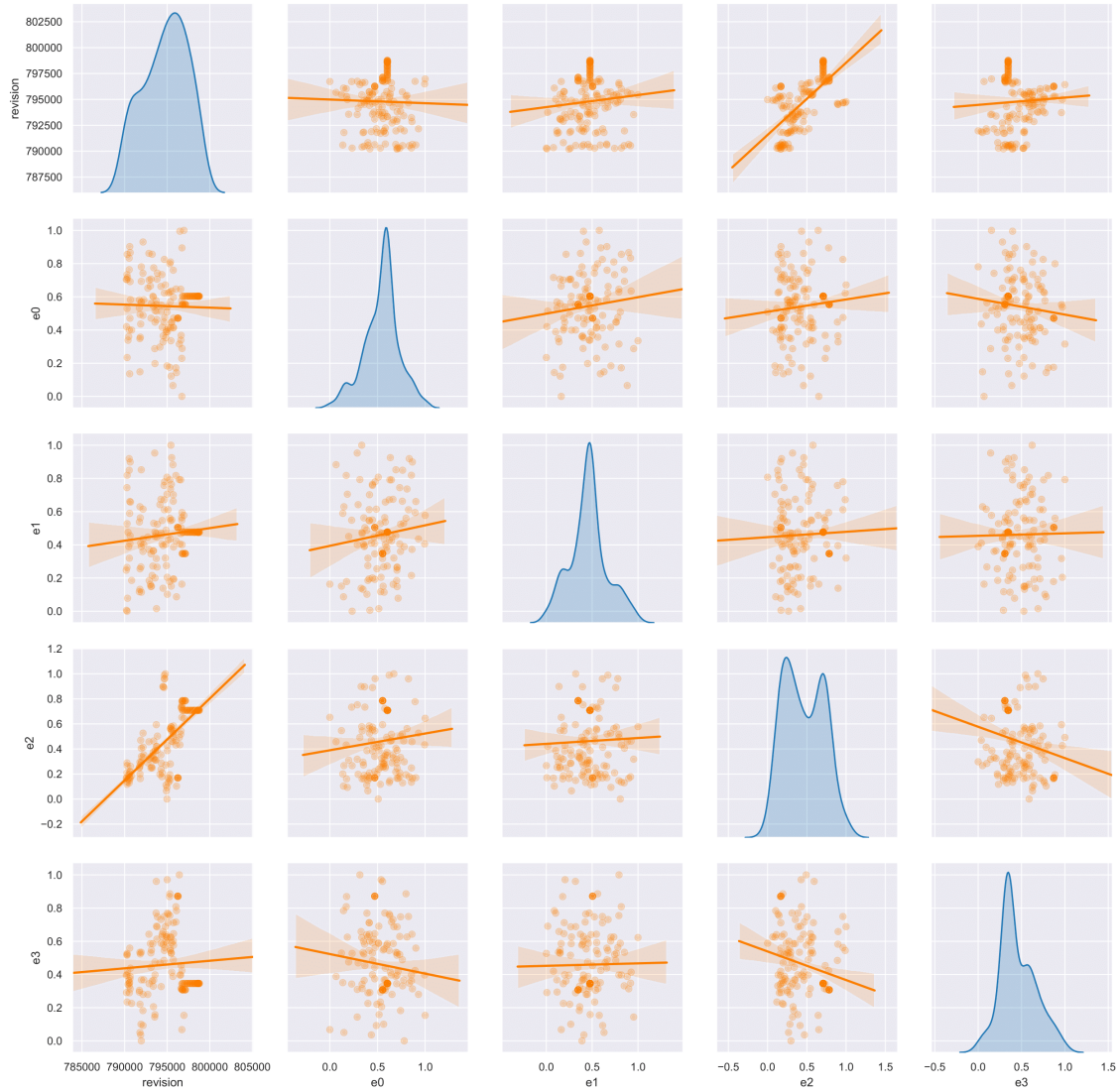

Figure 50: Event-level dataset scatter plot matrix with simple linear regression in test A.

.4

Performance measurements scatter matrix, test B

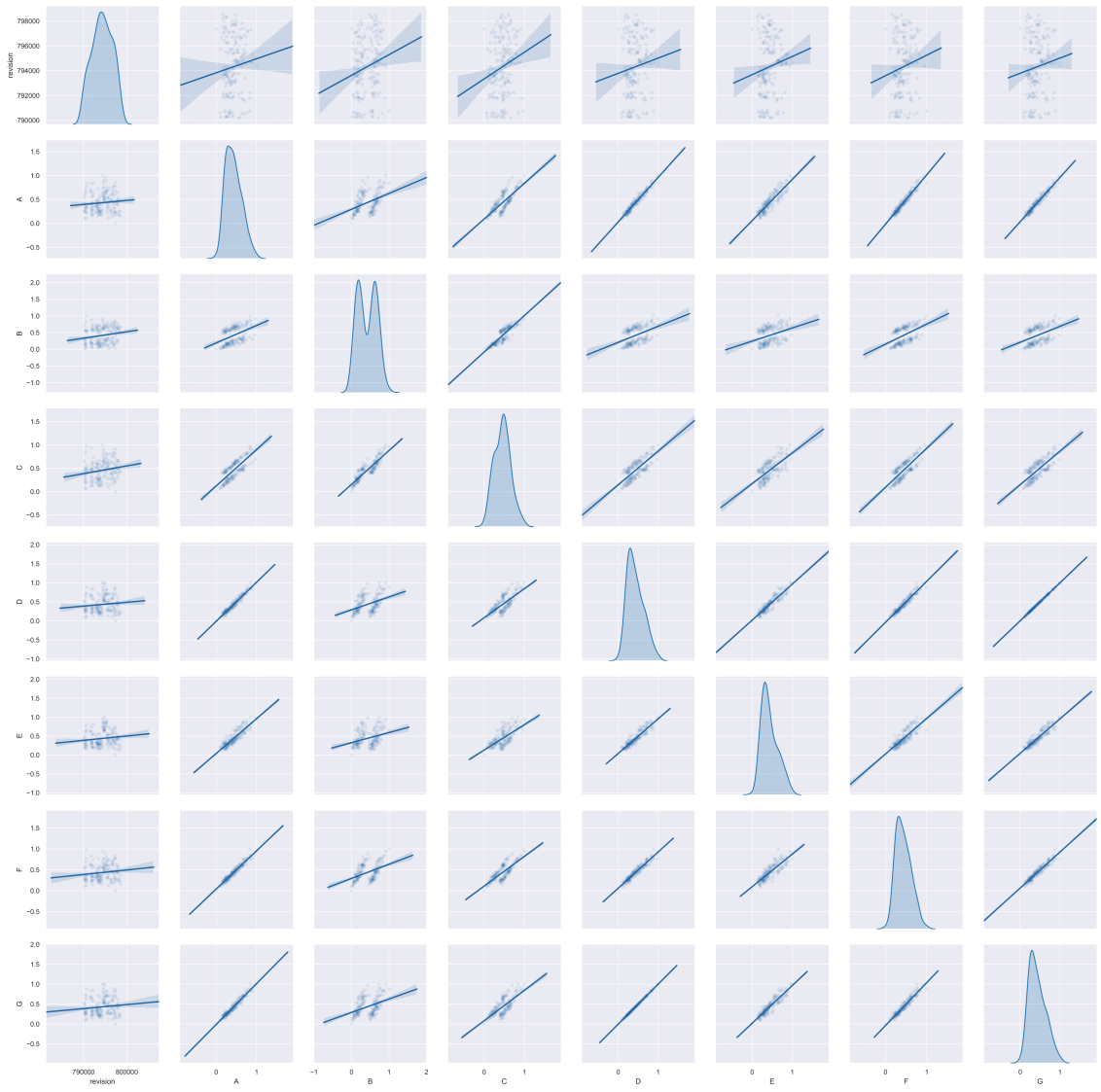

Figure 51: First seven PM scatter plot matrix with simple linear regression in test B.

The first seven PM are all highly correlated with some particular behavior in the PM B, it behaves as it has some threshold that triggers a change in its value, this could be similar to the behavior of a car gearbox

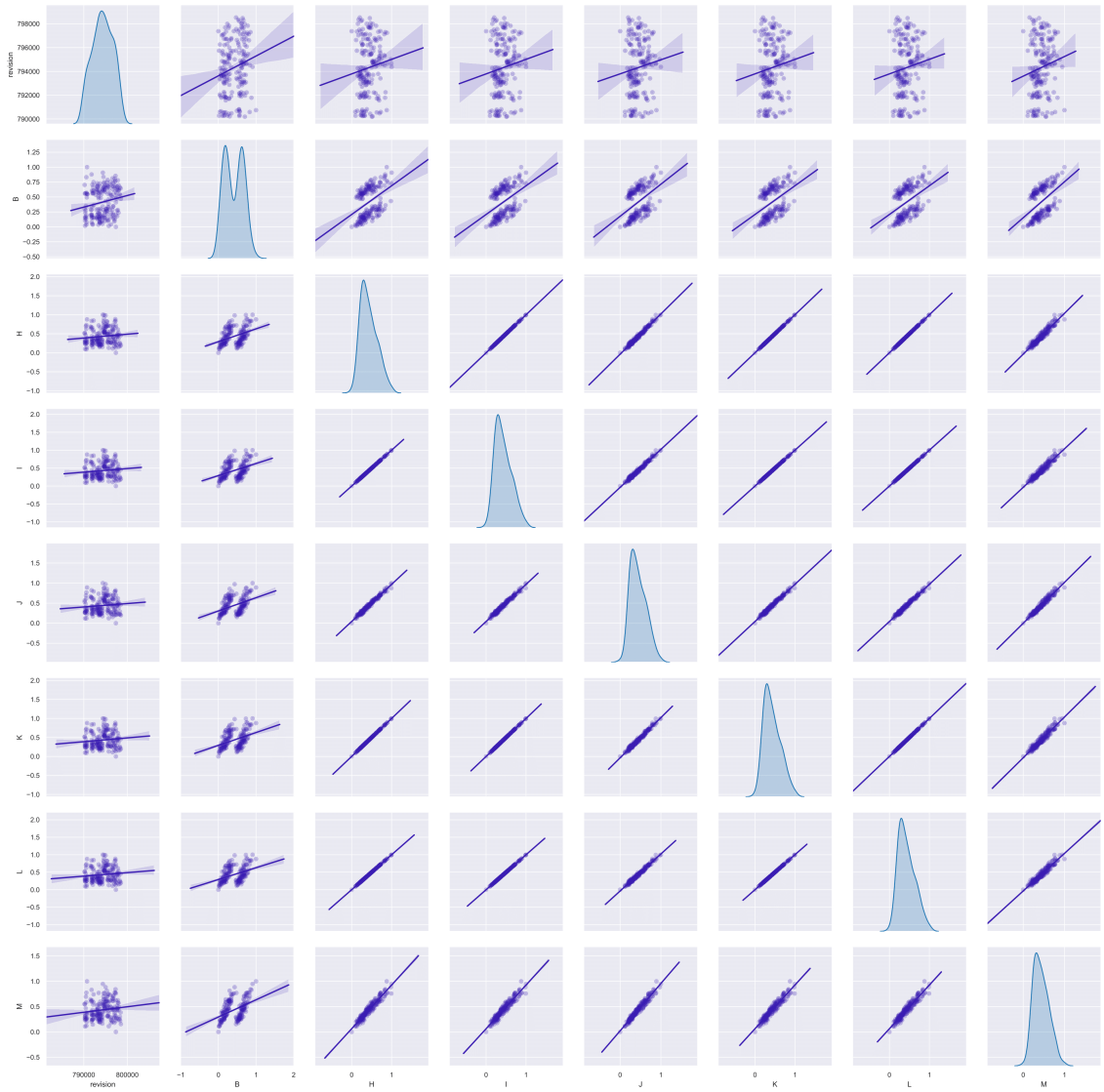

Figure 52: Last six PM scatter plot matrix with simple linear regression in test B.

Almost all performance measurements are highly correlated, one could conclude that they are actually measuring the same property of the system.

.5

Function-level scatter matrix by processor event, test B

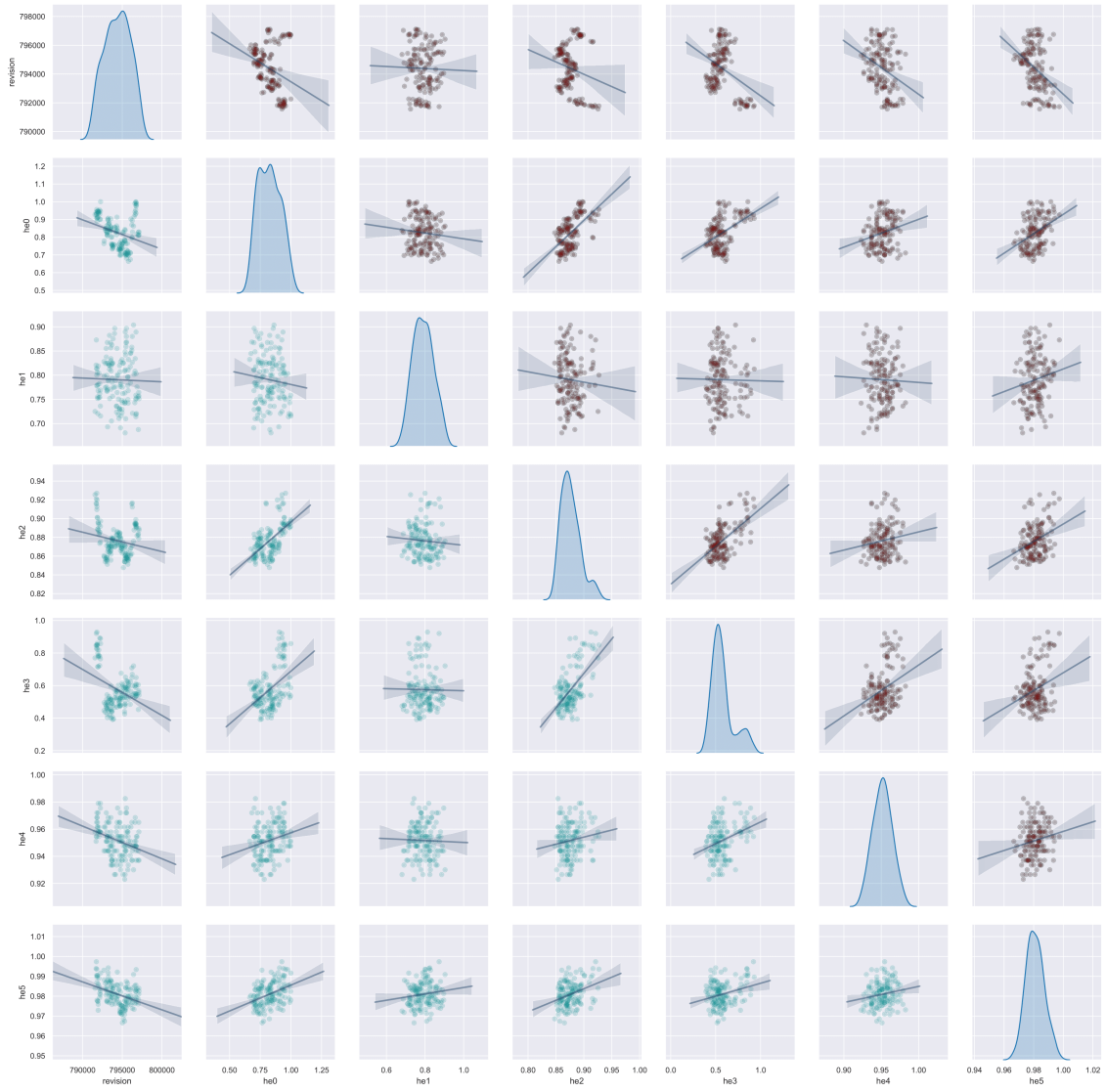

Figure 53: Function-level dataset scatter plot matrix with simple linear regression in test B.

.6

Event-level scatter matrix by selected events, test B

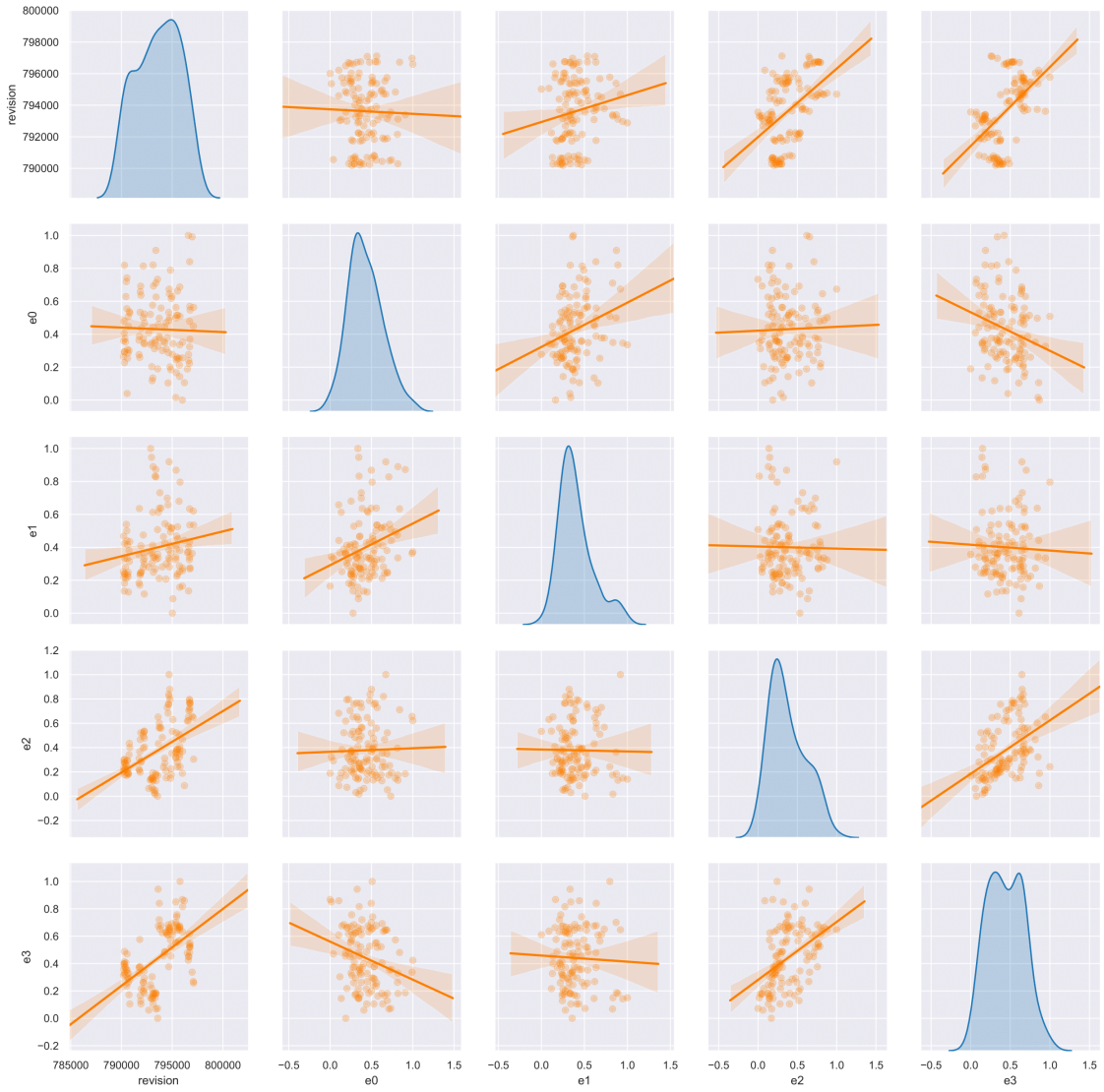

Figure 54: Event-level dataset scatter plot matrix with simple linear regression in test B.

.7

Selected features correlation heat-map for $F3$, test A

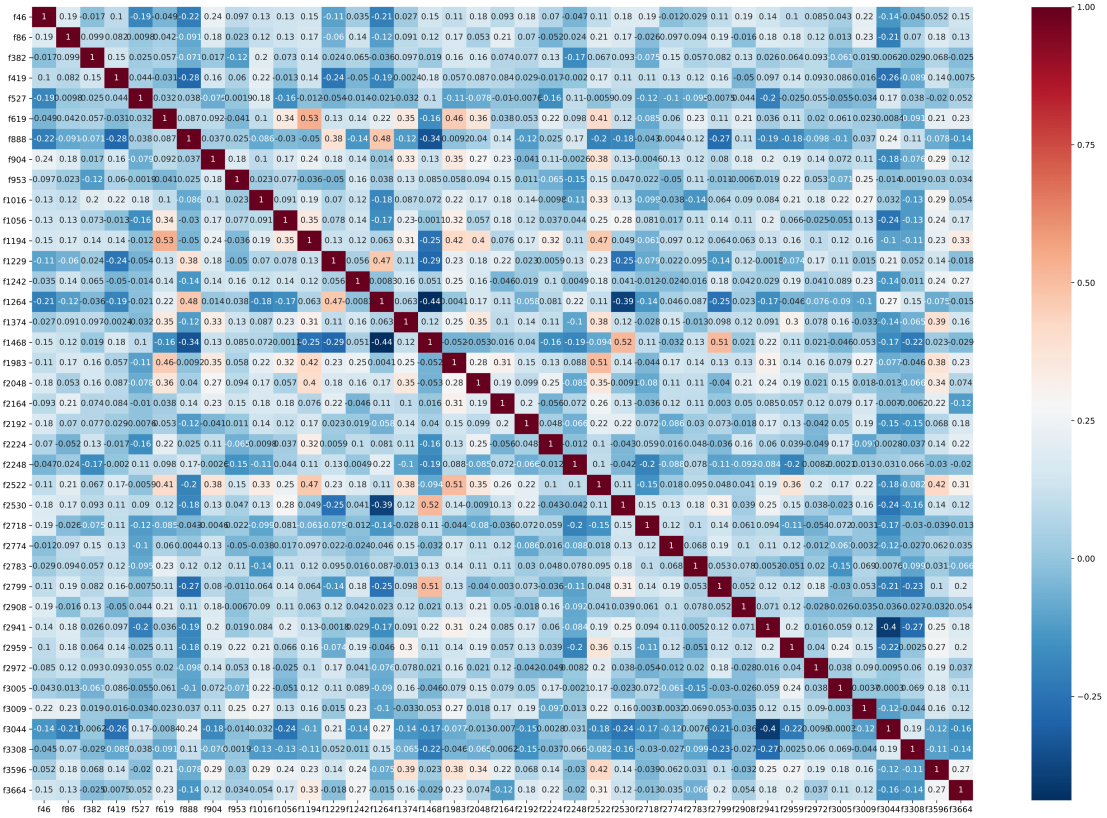

Figure 55: Selected features correlation heat-map for $F3$ in test A.

.8

Selected features correlation heat-map for $F3 + E2$, test B

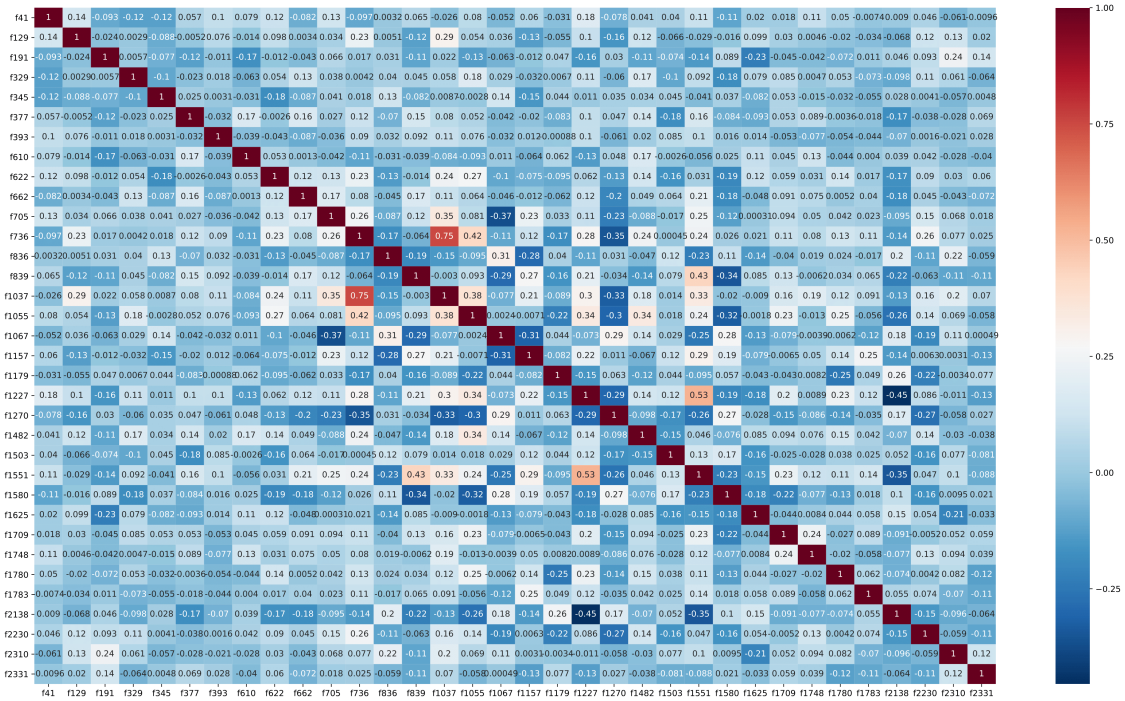

Figure 56: Selected features correlation heat-map for $F3 + E2$ in test B.