Machine Learning in Telecommunication Networks is Happening Now

Check how the experts do it

NOKIA

Book Title:
Machine Learning in Telecommunication Networks is Happening Now. Check How the Experts Do It.

Chief Editor:
Krzysztof Waściński

Content Supervisors:
Krzysztof Waściński, Sławomir Andrzejewski, Łukasz Pętlicki, Ireneusz Jabłoński, Alina Sikora-Godlewska, Witold Pawlus, Maciej Kondrat, Paweł Skoliński, Karolina Grześkowiak, Damian Czaja, Krzysztof Zieliński, Daniel Woźniak, Michal Pawlik, Wojciech Penar, Maciej Chyrc, Rafał Pasek, Wiktor Sędkowski, Daniel Woźniak, Ewa Boryczka, Weronika Białecka, Tomasz Przerwa, Adam Jankowiak, Mateusz Sołtysik, Alicja Figas

Language check:
Paulina Drewniak, Małgorzata Dryjańska, Katarzyna Niemira, Malwina Wrzosek-Bednarska, Weronika Sielicka

Editor:
Jarosław Danielak / Maciekkolabrandy Sp. z o.o.

Publisher:
Nokia Solutions and Networks Sp. z o.o.

We would like to acknowledge the help of all the people involved in this project. Without their support, this book would not have become a reality.

Dear Readers,

Knowledge sharing mission on telecommunication's global trends is extremely important matter for our company. That is why I am delighted to present you the fifth part of the Nokia Book.

This time on 144 pages Nokia's experts shared their wide knowledge and valuable experience about Machine Learning and Artificial Intelligence concepts and their applications in telecommunication networks and Nokia processes.

In Nokia we focus on future technologies creation and ML&AI development is essential to stay on the right track. This would not be possible without our colleagues experienced in data science or mathematics. This knowledge is directly applied to our products and this is the reason why we introduced this Nokia Book edition dedicated purely to ML&AI.

Authors have shown what are the important ML algorithms to consider, selected real case-studies from Nokia projects and finally various ML software development aspects which are prerequisite to build effective ML&AI solutions. Without a doubt this branch of science impacts on our life now and will do this in the future as well.

Book is recommended for both enthusiasts of telecommunication and AI&ML so they can discover how future of telecommunication will look like.

I would like to thank the authors for this journey into the future, for their effort and for great involvement in that project realization.

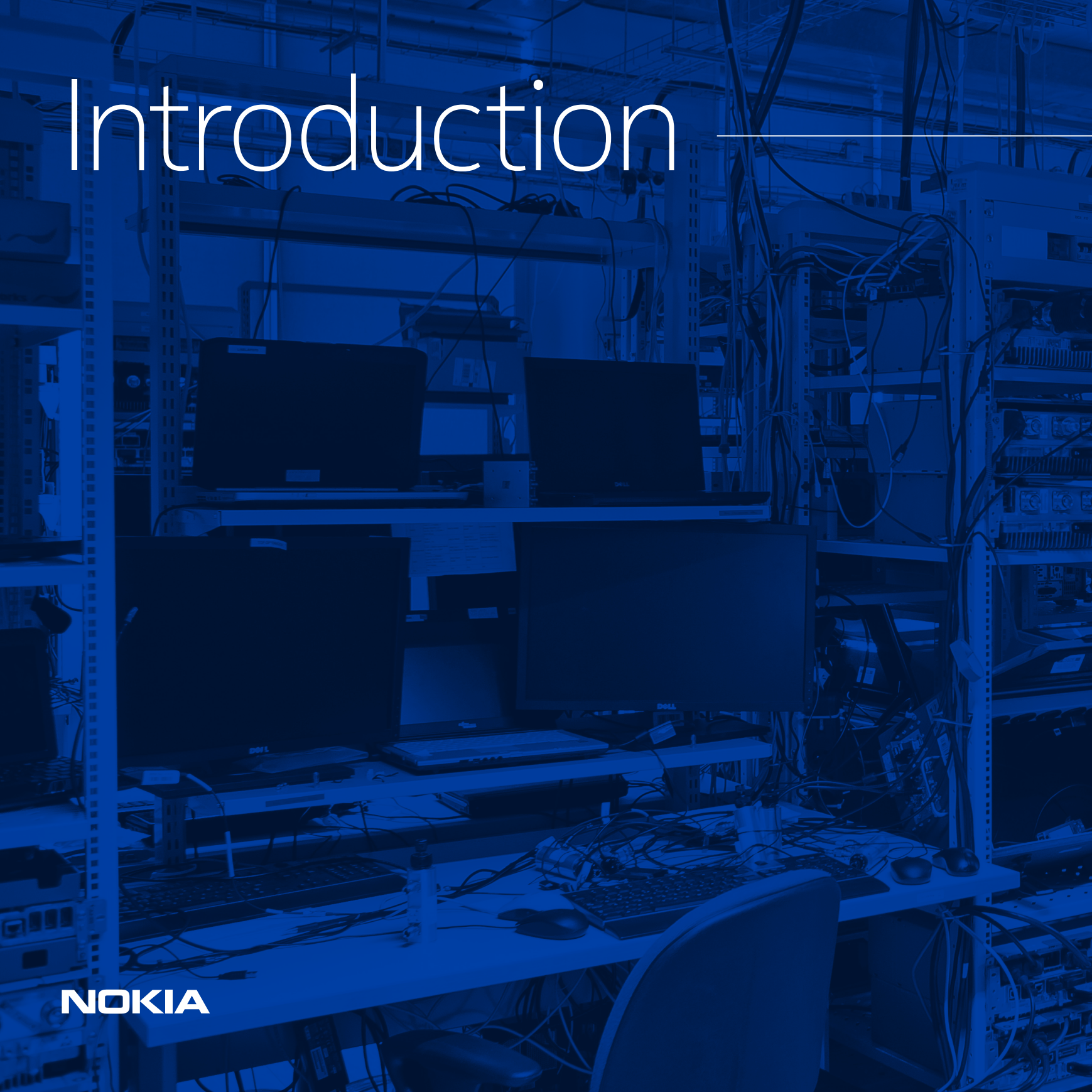
I wish you a pleasant read,

Bartosz Ciepluch

Head of Nokia Networks European Software
and Engineering Center in Wroclaw


Introduction
Selected Algorithms and Techniques
Case studies
Software Development Aspects

Introduction



NOKIA

1.1

Krzysztof Waściński
Nokia Book – Artificial Intelligence and Machine Learning. Introduction

Nokia Book – Artificial Intelligence and Machine Learning. Introduction

Krzysztof Waściński
Product Manager, Machine Learning Guild master
GS CS Network Engineering

NOKIA

Undoubtedly, Artificial Intelligence powered by Machine Learning has become in recent years a global megatrend. Media coverage and public discussion about it became inescapable. Many companies introduce products and services which utilize this technology. What is more, machine learning is so pervasive today that you probably use it dozens of times a day without knowing it. Why is that, and why should you care? Before we jump into the other articles in this book delving into intricacies of how it works and where it is applied, this introduction will give the necessary background. It provides you with a basic understanding of the motivation, importance and business impact of this technology, as well as its impact on Nokia's business and strategy.

1. Machine Learning – what it is and why it matters

The goal of this chapter is to provide an intuition on why Machine Learning has become so topical now, what it truly is and why it is important.

> *"AI is going to have as big impact on our society as electricity"*
>
> Risto Siilasmaa,
> Chairman of board, Nokia

> *"We are at the beginning of a technological revolution that will fundamentally alter the way we interact with each other, our working environment, companies, brands and governments. In its scale, scope, and complexity, the transformation will be unprecedented.",*
>
> Rajeev Suri, Nokia CEO

1.1 (R)evolutions

Over the last centuries, the recent generations have witnessed several industrial revolutions: starting with steam engine and iron casting in the first stage, through the second era of electricity, chemical engineering and telecommunications, to the 3rd one which fuelled social development with wide access to information through internet and mobile devices [1]. The last event in this series has a truly significant impact on today's world since it started the process of information digitalization and brought about the IT era. Different chunks of information existing only in physical world started to be transformed: from books on paper to files on a computer, digital maps and many more. Firstly, that enabled computers to assist human beings in centralized search tasks, massively boosting time-to-reach the right information across many sources. Secondly, every time such digitalization happened, it generated a massive amount of data. That is exactly what is AI revolution feeds on, since Machine Learning requires massive data [2].

It is more than expected that the amount of information we have available will grow rapidly in the next decades thanks to next generation of networks able to transmit more and more bits (**Figure** 1). Together with further improvements in silicon chips and storage capabilities, we can expect further acceleration of AI commercialization across all industries.

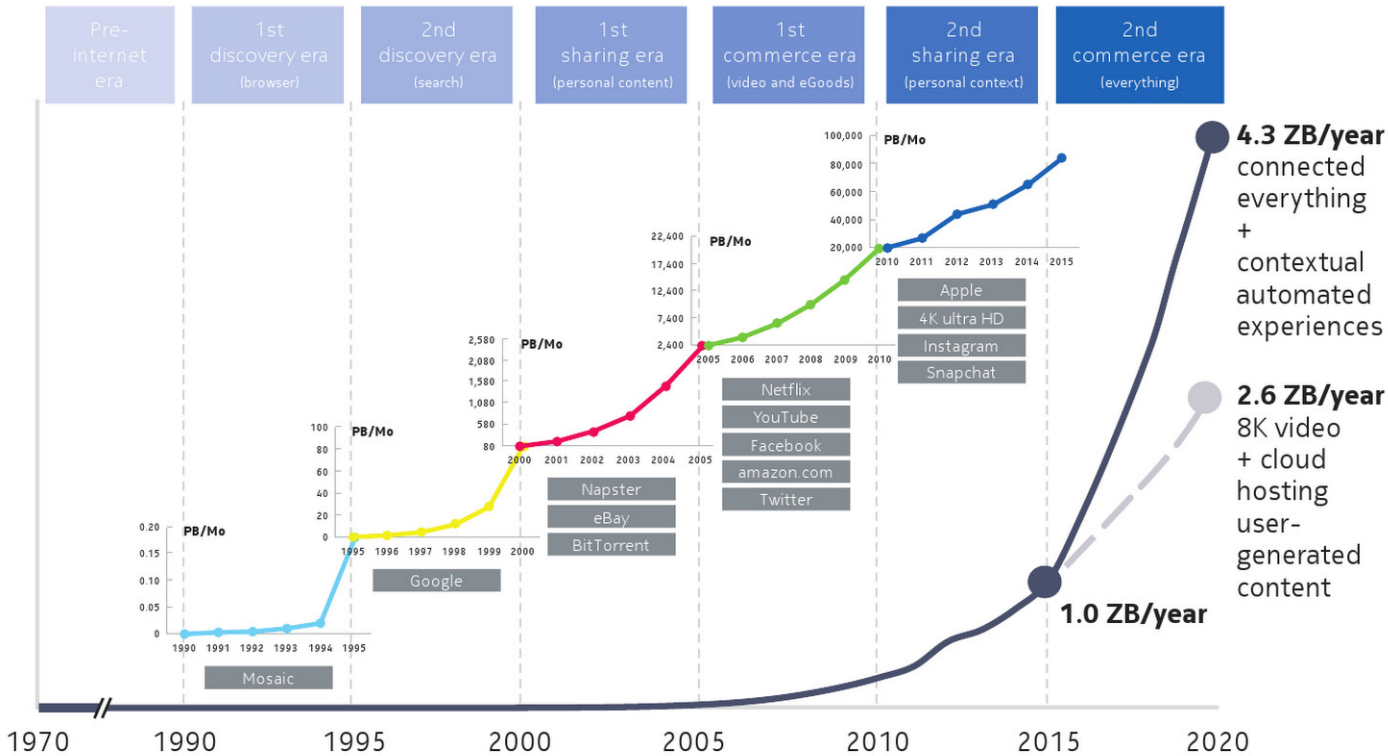
Learning is a time-consuming activity. That's no different for neural networks mimicking the human brain. Computer-based decision-making systems require so much of this data to reach acceptable accuracy. "Acceptable" is dependent on the product and its tolerance to errors: for some of them it might be just 1% (medical images, i.e. cancer recognition), for others a 20% margin is satisfactory (i.e. a recommendation system for movies will be less strict). Quite often is it also compared to human accuracy, and once computers are making fewer errors, the products gain trust. However, usually every new data-point for training matters and methods like transfer learning and synthetic data generation help to get that data in the large and diversified volume.

To make some analogy to ourselves: imagine how many times you tried to learn some specific task like juggling a ball. Each of the attempts can be thought of as an experiment with various parameters: strength, angle or speed. Until you've learned how to master this skill, you most probably evaluated millions of different combinations of those parameters. In other words, a lot of data has been used to train your muscles to perform this task. The same happens in unsupervised machine learning (you'll find more details in further chapters in this book). The only difference is that computers understand only the binary "0" or "1" information, so you need to properly describe the physical world with those.

1.2 Some history

Machine learning is not a new science – but one that has gained fresh momentum (see **Figure** 2). Its methods and algorithms, built upon mathematical and statistical groundwork, have been around for a long time, starting in 1950s. Researchers wanted to see if computers could learn without being explicitly programmed to perform specific, simple tasks. At that time, this was the stuff of science fiction, and soon it was put into the drawer for many years (called the "AI winters"). The current rebirth is thanks to the recent advancements in computer science, offering high processing power which makes it possible to automatically apply complex mathematical calculations to big data – over and over, faster and faster. In just half a century, what was once theory becomes real.

Figure 1 Analysis of the growth in core network traffic (dark curve) since the dawn of the internet era in terms of the constituent five-year trend segments (data shown with expanded scales) (Bell Labs Consulting [3])

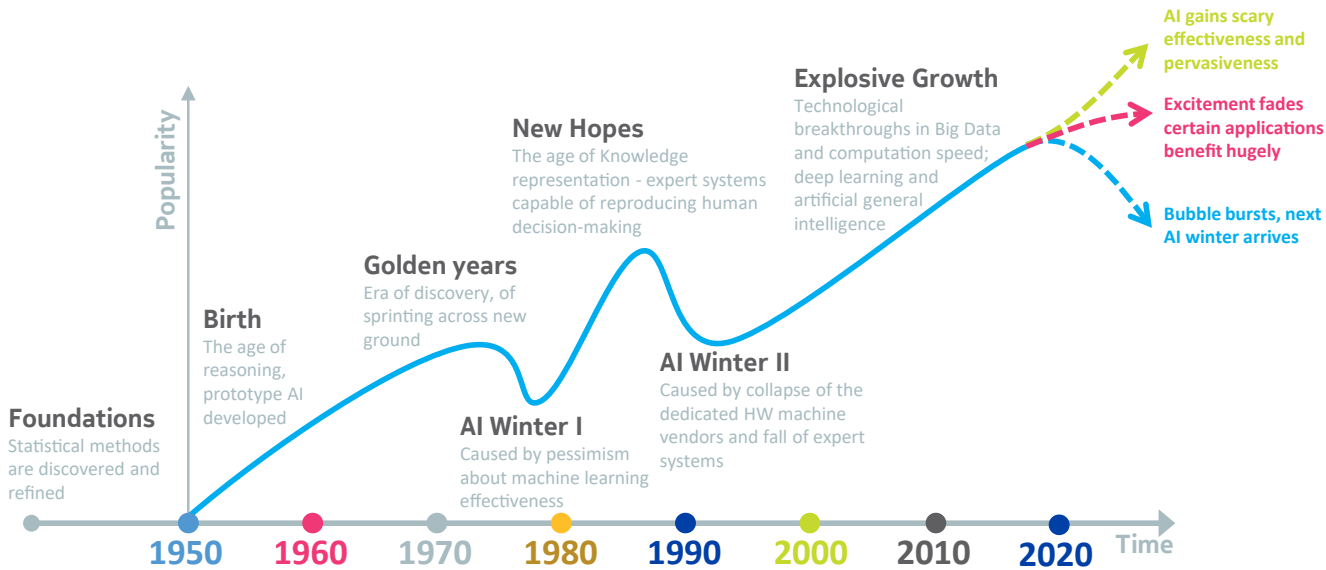


Nowadays we see a wide variety of AI applications in our lives and across industries. The value of machine learning technology has been recognized especially in organizations working with large amounts of data. By mining insights from their data, those organizations can work more efficiently, gain an advantage over competitors or introduce new innovative services. Just a few examples to illustrate the importance of machine learning:

- Self-driving cars (i.e. Google, Uber, Tesla),
- Object recognition in images or videos (i.e. Facebook, Instagram),
- Content recommendations systems (i.e. Netflix, HBO),
- Intelligent service chatbots that understand speech (SIRI, Google Now, Cortana),
- More robust fraud detection systems,
- Healthcare: robotic surgeries and automatic retinal images analysis

How far this technology can take us remains to be seen (**Figure 2**). Amplification of human intelligence with artificial intelligence has the potential of helping civilization flourish like never before and start the fourth industrial revolution [1]. On the other hand, we might see a next AI winter, if the effectiveness will disappoint' us. There's also a risk that AI would learn from the dark side of our nature – a topic fuelling many Sci-Fi movies and scaring many people (recall Microsoft Tay chatbot going rogue [6]). Although it is very unlikely that machines will exhibit broadly-applicable intelligence comparable to or exceeding that of humans in the next twenty years, it is to be expected that machines will reach and exceed human performance on more and more tasks, and we will see the excitement turn into a productivity increase.

Figure 2 A brief history of artificial intelligence (based on [4] among others, [5])



1.3 Principles

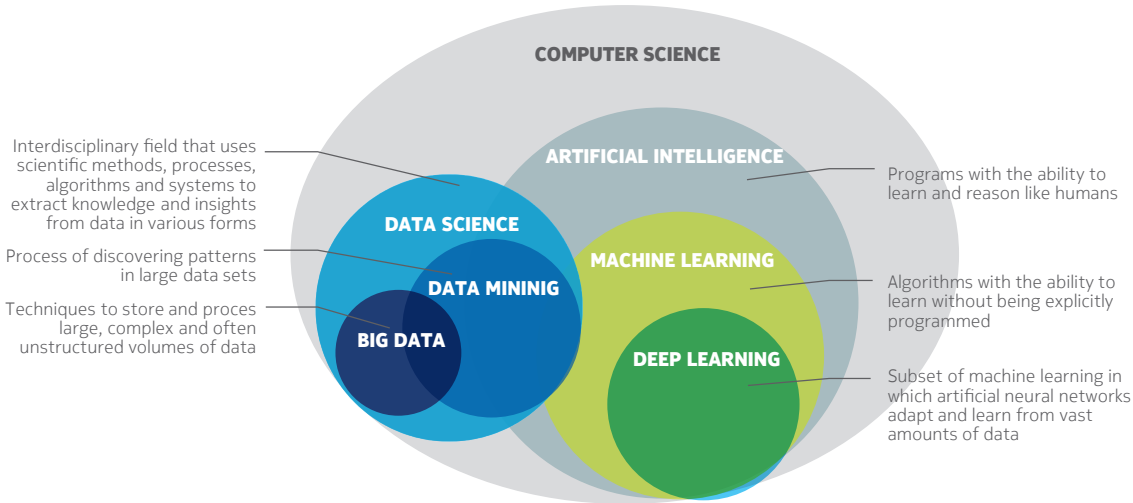
Machine Learning and Artificial Intelligence mean different things to different people. These and other terms in this field of computer science have confused many; they are often used interchangeably. For some people, AI is about machines that can match or surpass our intelligence at every task. For others, it is almost any data processing technology or "fancy things computers can't do yet" (but once they can it is not fancy AI anymore). Machine Learning, on the other hand, is often associated with its non-fancy and unsuccessful applications which caused the AI winters. To set the scene for further articles in this book let me explain **Figure 3**, which should help to properly clarify and delineate those and other closely related terms and fields.

Firstly, Artificial Intelligence is a broader umbrella under which Machine Learning and Deep Learning reside. AI should be then a computer program with human abilities: learning, reasoning, sensing, creativity, planning, identifying patterns and making decisions like we do. To be more precise, this kind of AI is called the Artificial General Intelligence (AGI). In fact, although it can already automatically perform many functions in some narrow fields, today's AI is still extremely limited compared to general human intelligence. Humans are stronger in the absence of data, or when applying learnings from multiple fields is necessary, for example analysis of the best strategy or research (e.g. What are gravitational waves?). On the other hand, humans are rarely quick when the amount of data is high, for example in cases of fraudulent operations discovery or financial risks calculations.

Machine Learning is a branch of AI based on the idea that systems can learn from data. ML utilizes data science and statistical techniques to allow machines to automatically identify patterns and build models to make decisions without human intervention (without being explicitly programmed). Data science, in turn, is a general, interdisciplinary field related to scientific methods of knowledge extraction from various forms of data. However, in the context of ML it plays a crucial supporting role. Without advanced Big Data technologies and exploration (data mining) techniques, ML would not be as successful as it is today. It is Data Science that gave the foundation for building ML systems as they are today.

Lastly, Deep Learning is basically Machine Learning on steroids. It employs Neural Networks, a structure inspired by the human brain, namely the interconnecting of many neurons. Neural Networks (NN) algorithms form multiple layers, processing the data and making decisions. Deep Learning is simply Neural Networks with many hidden layers of decisions – each of them extracting some different features of the input data. For example, a first layer of face images processing could detect the edges, the second the shapes and third

Figure 3 Artificial Intelligence terms



one could merge previous outcomes to distinguish the actual parts of the face like nose or eyes. What each layer does is often not so easy to explain to a human – but it was found that NNs with a lot of such abstraction layers are able to beat human cognition capabilities. Thanks to those Deep Neural Networks, ML has recently attracted so much attention, thus starting the next "AI spring".

2. AI in telecommunication networks

Nokia has the ambition to become the leader in applying AI to its products and services to boost the evolution of the telecommunication industry. We are uniquely positioned to pursue this goal, having access to massive amount of network data, hardware and domain expertise.

2.1 The need

Although at first glance the most cited applications of AI are in the consumer market, it is expected that machine learning will become pervasive in the Telecom Industry as well. The motivation behind it is the introduction of next generation networks, which will significantly increase the network operational complexity. There will be more parameters and dimensions to optimize in the new network topologies, and network elements redefined through cloud evolution. Additionally, the pressure on cost per Gbit requires improvements in HW performance, thus driving the invention of improved algorithms consuming less CPU power. The operators also demand better efficiency, agility and DevOps capabilities. New network topologies redefine the role of network elements.

In general, there seems to be a common vision that new 5G networks need to be re-thought and rebuilt with AI in mind to solve these issues.

Nokia Bell Labs already created an AI-ready Future X network architecture ([3]) which supports the capture, transmission, storage and processing of massive data sets from network elements. Those architectural blueprints are being embedded in the HW processing and memory requirements and key algorithms that drive artificial intelligence supported by the Nokia network equipment.

The next chapter summarizes the main areas where these forms of artificial intelligence could be applied. Since literature for this subject is quite wide, I'd like to provide the high-level overview and go through selected use-cases.

2.2 Data sources

To enable ML based solutions, the necessary data must be properly prepared and provided for model development and training purposes. Radio Access Networks (RANs) are data-rich environments, where data is continuously gathered in the form of radio measurements, configuration files, traces or other system observations by

thousands of user devices and network entities. These could be distinguished by the level of where observations are taken: at the network elements (base stations, routers etc.) and at the end-user devices (UEs). A summary of those data sources is presented in the table below. Most network optimization tasks utilize those data sources and could be further enhanced with ML-based applications.

Table 1 Mobile network data sources

Level	Source	Information
Network equipment data	Configuration Management (CM)	equipment locations, parametrization, feature activation, HW configuration, power settings, spectrum etc.
	Performance Management (PM)	radio measurements performance counters and resulting Key Performance Indicators (KPIs): throughput, QoE, delays, procedures success ratios etc.)
	Failure Management (FM)	information on alarms raised at the network element
	Per Call Measurement Data (PCMD)	performance, QoE, traffic and other parametrisation data collected per session
	BTS traces	deep log information collecting detailed information on the equipment operation
User equipment data	Device information	type, manufacturer, MAC address, user settings, personal information etc.
	Carrier (SIM) information	operator name, connected cell identifiers (i.e. eCGI, BTS ID)
	Profile: Sensors	temperature, gyroscope (i.e. speed), GPS, etc.
	Performance	established data bearers, their throughput, latency, received signal power, block error rate
	System logs	software and hardware failure logs etc.

Nevertheless, it should be noted that this taxonomy does not include the data which is internally available and being processed in the equipment, e.g. signal processing and internal databases. They can be very specific depending on the type of equipment, but the next chapter shows that they have huge potential for AI as well.

Finally, collection and provisioning of such large volumes of mobile data might pose significant challenges. That aspect should also be considered to enable ML use cases.

2.3 Major applications

Although there's no single disruptive AI application in communication networks, the augmentation is already underway, and many ML methods are already being embedded into network procedures, tools and processes. In mobile networks so far, analytics has been an afterthought, being applied on a case by case basis as a bolt-on. If we focus more on RAN, the most promising categories of use cases for AI are in fact the same as the ones in mobile networks in general:

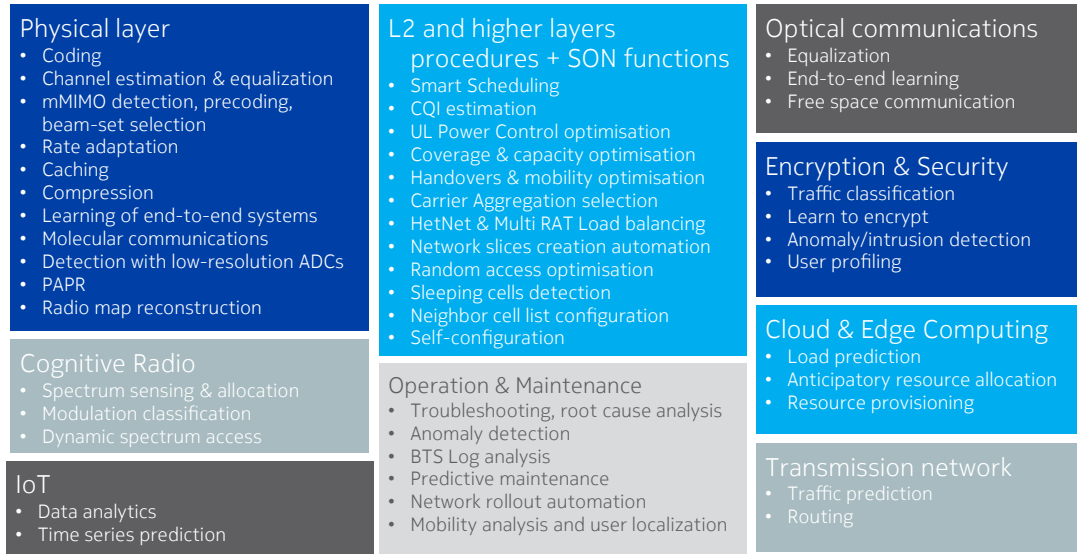
- Predictive and preventive performance, capacity and fault management.
- Cognitive and self-optimizing networks SON) products
- Traffic management based on real-time customer experience
- Security and privacy management
- Hosting 3rd party edge computing applications

In general, machine learning could become a useful tool for improving the performance of 5G networks, particularly in areas requiring high complexity solutions to deliver desired performance.

Comprehensive surveys of the applications have already been conducted (i.e. [7], [8], [9]) with hundreds publication references. It shows that AI can be used to make small improvements in many areas, reducing the overall complexity & latency. **Figure 4** summarizes selected AI application areas, especially in the radio base station algorithms and SON functions. These are enumerated to better illustrate the art of the possible, but not meant to be a comprehensive or exhaustive listing. Throughout the rest of this chapter we'll discuss some of them.

The first area of base station where ML could be implemented is the physical radio layer (L1). Most of the processing which happens inside that layer is related to signal processing, coding, channel estimation, and beamforming calculations. These functions' logic is currently designed based on solid mathematical statistical foundations and information theory, but with stationary assumption relying on Gaussian distribution. A practical system, however, has many imperfections and non-linearities that can only be approximately captured by such models. Timothy J. O'Shea and Jakob Hoydis[10] try to address this problem with deep learning methods. The authors propose to model a communication network as an end-to-end reconstruction optimization task, where the transmitter and receiver

Figure 4 Selected areas of ML application in mobile networks



will jointly learn its implementations. That would mean that it would not need to optimize for any specific hardware or channel model. The results prove that in theory it could yield performance close to state-of-the-art channel models, but that practical implementation would still need to deal with the unambiguity of channel transfer functions and exponential training complexity due to continuous transmissions.

Another example of ML application in physical layer would be digital beamforming pattern and antenna weights optimization in a massive MIMO (mMIMO) systems in 5G. Beamforming brings a new dimension where various beam sets may be applied for the cells depending on the environment or traffic distribution (see **Figure 5**). Narrow beams can be used in the regions of high user density whereas wider beams could be used in the regions of low user density. Based on the channel sensing measurements, an optimal beamformer configuration might be derived, which can improve the user throughput significantly. However, it depends on several factors, such as the topology of the cell, physical environment, distributions of users, etc. Manual optimization or conventional algorithms-based implementation result in large gaps to optimal performance in these complex scenarios. Machine learning excels in delivering near-optimal performance with reasonable implementation complexity.

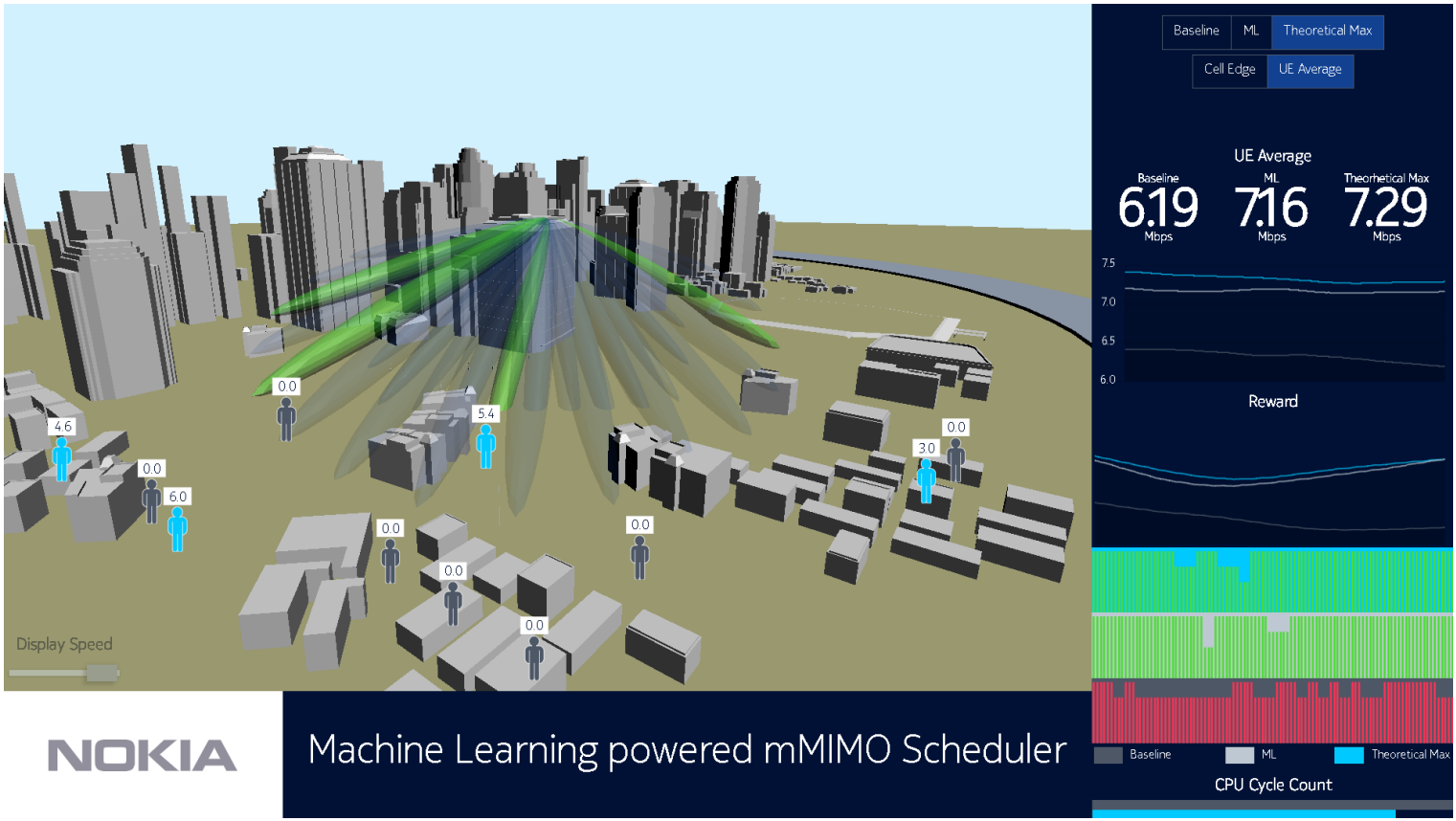
ML would also be essential in management of higher layers (L2-MAC and others above it). Within this part of the base station reside the procedures managing the user and control plane resources, which are critical for network performance. Mobile operators have always been looking for a way to improve the network performance and their users experience. For that reason, further inventions in Radio Resources Management (RRM) procedures ([11]) helping to get the throughput higher and latency smaller capture an immediate interest within the industry. That results in further inventions, for example:

- scheduler ([12]),
- interference coordination for UL power control ([13]),
- HetNet interference coordination ([14]),
- mobility management & handover optimization ([15]),
- CQI estimation and optimization, random access or sleeping cells detection.

Besides performance, ML could also help to cut down operational efforts and provide cost savings. The key concept to do this is realized in the so-called SON paradigm (Self-Organized Networks), which will optimize L2 procedures automatically. In general, SON functions can be divided into the self-configuration, self-optimization and self-healing areas. They have been already verified and embedded in LTE networks. Out of the variety of areas listed in **Figure 4**, it is the SON use-cases that have drawn a lot of attention in research ([7], [8], [9], [16]) promising to help manage 5G networks complexity.

The last use case worth mentioning is Indoor positioning using RF fingerprint and Neural Networks (**Figure 6**). UE localization can be a very valuable tool for network optimization, particularly in the context of proactive management of the network (i.e. proactive

Figure 5 Nokia ML powered mMIMO scheduler 2 demo

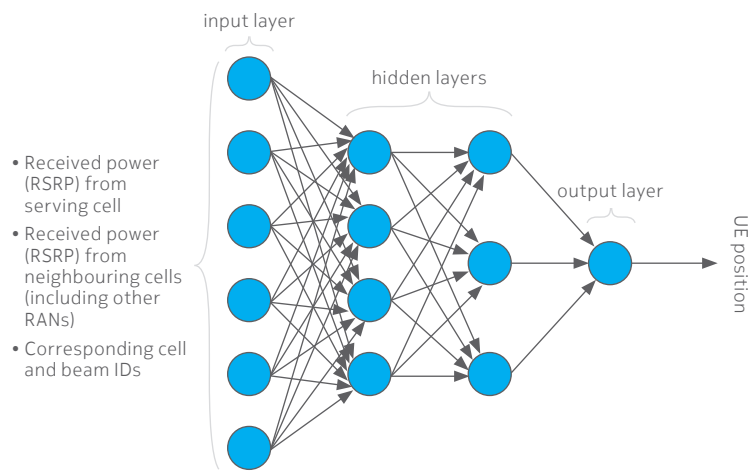


handover to avoid blocking effects at mmWave bands). Unfortunately, it is often unavailable if the user switched off GPS or remains inside buildings were GPS signal is lost. Nokia internal studies have shown that high position accuracy information can be provided by utilizing existing wireless/cellular networks radio measurements paired with neural networks.

2.4 Other areas

Besides pure telecommunication, there in a large variety of use cases in the other domains, applicable for Nokia. One of them is application of Natural Language Processing in the customer support aiding ticket handling process (routing) or automated logs parsing and analysis. They would boost operational efficiency and shorten the time required for the ticket resolution. If further enhanced with intelligent virtual assistants (or chatbots), they could enable new ways of customer interaction. AI is required there to shape the

Figure 6 Deep Learning-based UE positioning concept.



conversation, so that the other side will easily get the information via human-like chat experience. Such assistants could retrieve the information from internal knowledge engineering systems (you can read about it in one of our articles in this book) and efficiently offload the customer support staff.

Aside from that, there are plenty of internal processes, including actual product development (implementation, code review, testing, bug fixing), finance, operations and reporting activities where AI already finds ways to improve efficiency.

3. Summary

While it is still a subject of ethical, economic and political debates, AI already brought enormous benefits in lifestyle, entertainment, and even saved people's lives. It is also driving tremendous economic value, maybe even in the billions. It is certain that its full potential is still undiscovered. Its first embodiments emerge in telecommunications. With current investments in this field, AI will quickly make rapid progress to rival human performance. It will open many new possibilities to deal with the problems of mobile networks and our planet in general. That's why we introduce this book: to spark your curiosity and bring the next exciting AI breakthroughs.

References

[1] N. Rajeev Suri, "The Fourth Industrial Revolution," MWC, 2018. [Online]. Available: https://www.mobileworldlive.com/on-stage/mwc/keynote-8-the-fourth-industrial-revolution-rajeev-suri/.

[2] R. Siilasmaa, "Risto Siilasmaa on Artificial Intelligence/Machine Learning," [Online]. Available: https://www.youtube.com/watch?v=KNMy7NCQDgk.

[3] M. K. Weldon, The Future X Network: A Bell Labs Perspective, Taylor & Francis Group, 2016.

[4] T. F. F. Watch, "China's Digital Landscape and Rising Disruptors - Module 2.6 Artificial Intelligence," [Online]. Available: https://www.slideshare.net/futurewatch/future-watch-chinas-digital-landscape-and-rising-disruptors-module-26-artificial-intelligence.

[5] Wikipedia, "History of artificial intelligence," [Online]. Available: https://en.wikipedia.org/wiki/History_of_artificial_intelligence.

[6] S. Perez, "Microsoft silences its new A.I. bot Tay, after Twitter users teach it racism," TechCrunch, 2016. [Online]. Available: https://techcrunch.com/2016/03/24/microsoft-silences-its-new-a-i-bot-tay-after-twitter-users-teach-it-racism/?guccounter=1.

[7] C. Zhang, P. Patras and a. H. Haddadi, "Deep Learning in Mobile and Wireless Networking: A Survey," *IEEE COMMUNICATIONS SURVEYS & TUTORIALS,* 2018 .

[8] P. V. Klaine, M. A. Imran, O. Onireti and R. D. Souza, "Survey of Machine Learning Techniques Applied to Self Organizing Cellular Networks," *IEEE Communications Surveys & Tutorials,* 2017.

[9] Q. Mao, F. Hu and Q. Hao, "Deep Learning for Intelligent Wireless Networks: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials,* 2018.

[10] J. Hoydis and T. J. O'Shea, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communication Networks,* July 2017.

[11] F. D. Calabrese, L. Wang, E. Ghadimi, G. Peters, L. Hanzo and P. Soldati, "Learning Radio Resource Management in RANs: Framework, Opportunities, and Challenges," *IEEE Communications Magazine,* 2018.

[12] F. Pianese and e. al., "Optimized Data-driven MAC Schedulers for Low-Latency Downlink in LTE Networks," 2016.

[13] S. Deb and P. Monogioudis, "Learning-Based Uplink Interference Management in 4G LTE Cellular Systems," *IEEE/ACM Transactions on Networking ,* April 2015.

[14] P. Monogioudis and e. al., "Algorithms for Enhanced Inter-Cell Interference Coordination (eICIC) in LTE HetNets," *IEEE/ACM Transactions on Networking ,* Feb 2014.

[15] Z. Ali, N. Baldo, J. Mangues-Bafalluy and L. Giupponi, "Machine Learning Based Handover Management for Improved QoE in LTE," in *IEEE/IFIP Network Operations and Management Symposium (NOMS 2016)*, 2016.

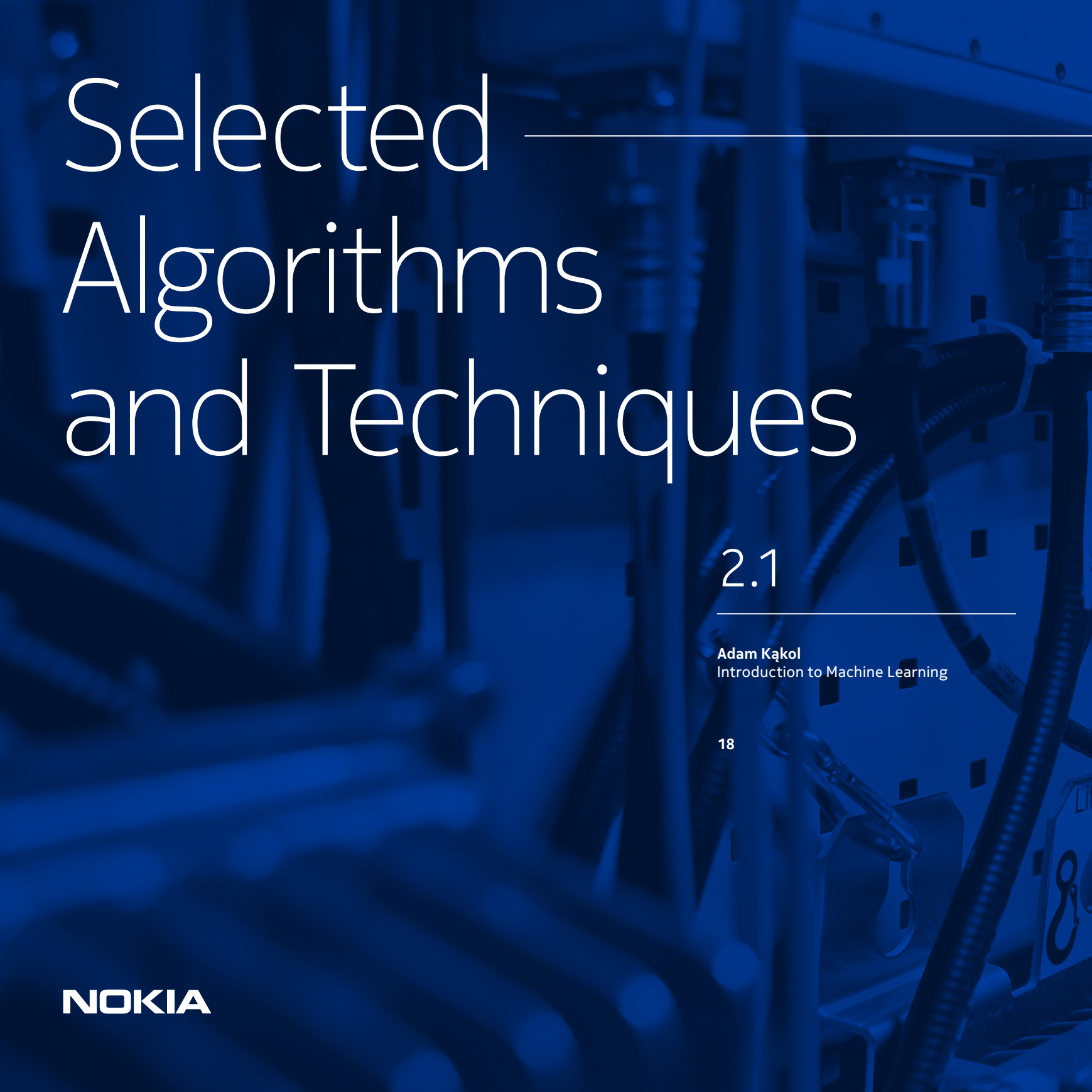
[16] G. F. Ciocarlie, C.-C. Cheng, C. Connolly, U. Lindqvist, K. Nitz, S. Novaczki, H. Sanneck and M. Naseer-ul-Islam, "SON Verification for Operational Cellular Networks," 2014.

[17] A. Ng, "Artificial Intelligence is the New Electricity," Medium, [Online]. Available: https://medium.com/syncedreview/artificial-intelligence-is-the-new-electricity-andrew-ng-cc132ea6264.

About the author

Tribe Product Manager responsible for managing backlog, prioritizing and guiding the work of the Network Engineering organization. Experienced telecommunication engineer with LTE and 5G background. Leading the execution of iconic customer projects in area of system-level simulations and network performance analytics. TC-Wroclaw Machine Learning Guild host and AI enthusiast.

Krzysztof Waściński
Product Manager, Machine Learning Guild master
GS CS Network Engineering



Selected Algorithms and Techniques

2.1

Adam Kąkol
Introduction to Machine Learning

18

NOKIA

Selected Algorithms And Techniques

Introduction to Machine Learning

Adam Kąkol
Function Squad Group Lead
MN CDS Customer Documentation

NOKIA

Machine Learning (ML) has conquered the market and become one of the hottest topics today. However, what does it mean for a computer system to learn something? The first two chapters of the article introduce the most important terms and classifications of machine learning algorithms, like supervised and unsupervised, online and offline, instance and model-based learning. The third chapter describes the approach to a ML project, starting from gathering and evaluating the training data, through selecting the right algorithm, to testing and fine-tuning it.

1. Definition of machine learning

Machine learning is one of the most important foundations of artificial intelligence. It consists of understanding data structures, finding patterns, and building conclusions. In traditional programming, a computer is given step by step, explicit instructions to solve a given problem. Machine learning introduces generic algorithms based on statistical techniques that can be fed with samples of data to find a solution. We say that a machine learns if the solution it generates improves progressively over time.

One of the well-known examples of machine learning use cases is spam recognition. It's a classification problem. The goal is to categorize an object using a fixed set of categories: spam or ham. A historical collection of such mails would be called a *training set*. Each of the elements of this set is a *training instance* or *training sample*. Machine learning algorithms analyze the training set to find patterns and relationships. After training, the machine can be tested as to whether it can correctly classify mails that were not part of the training set. The ratio of correctly identified spam messages is called the *algorithm's accuracy*.

Note that the initial algorithm doesn't have any predefined rules on how to detect spam. After training, we can look at the patterns identified by the machine. This is a summary of the input data and can be a subject to further analysis. This process is known as *data mining*. It means that machine learning can be used not only to make predictions, but also to understand the structure and to discover relationships in big collections of data. Data mining has great potential in business and it's used in many cases, such as performing market analysis to identify new product niches, finding the root causes of manufacturing problems, or preventing customer attrition, just to name a few.

2. Classification of algorithms

There are lots of machine learning algorithms, but each of them can be assigned to basic categories, depending on the:

- supervision model
- ability to learn incrementally
- way of making predictions.

To better understand which algorithm to choose to solve your problem, let's explore them in more detail.

2.1 Supervised, unsupervised, semi-supervised, and reinforcement learning

Supervised learning is one of the most frequently used machine learning techniques. The training set consists of both input parameters x and expected output y called *label*. The algorithm learns how to map them to make predictions.

$$y = f(x)$$

The name of this approach comes from the fact that the learning process can be compared to a teacher supervising a pupil. The teacher knows the correct answer. The algorithm iteratively makes predictions on the training data and is corrected by the "teacher". The learning process continues until the algorithm achieves an acceptable level of performance. It might happen that iterations are not needed when the learner has enough resources to achieve optimal performance without them.

Supervised learning is used for handwriting and speech recognition, spam filtering, face recognition, and many more.

Supervised learning can be further divided into:

- **Classification**, if the output belongs to a fixed set of values, for example, image categorization based on captured objects like cats or dogs.
- **Regression**, if the output is a continuous value, for example: stock price prediction.

In *unsupervised learning*, the training set is not labeled, or labels are identical. It means that the algorithm doesn't know what the expectations are. However, it can decide that data lives in two or more clusters. Google News applies unsupervised learning to group articles from different websites into logical categories. Marketers create customer segments to compress information about end users, while still maintaining the structure and usefulness of data (*dimensionality reduction*). IT uses unsupervised learning for anomaly detection to protect systems and discover security gaps.

Unsupervised learning can also be used for *association rules mining* to find interesting relationships and dependences in large sets of data.

Semi-supervised learning takes a data set which it labeled partially or not labeled at all. A labeled set is usually much smaller. Once clusters are identified through unsupervised learning, algorithm accuracy is further improved by using a labeled set. The most commonly known example of semi-supervised learning is tagging friends in photos. The algorithm identifies the same people in the gallery. When a human assigns a name to a person, the semi-supervised algorithm assigns the same name to that person on other photos as well.

In supervised learning, the training set comes with the expected answer. In *reinforcement learning*, there are no answers, and sometimes even no training sets. The *agent* itself must learn how to perform the task to maximize its long-term performance. Learning is realized by experimenting with the environment. The agent performs different actions and then receives either *rewards*, if an action was right, or *penalties*, if an action was wrong. Penalties are also known as *negative rewards*. Reinforcement learning is the method used with our pets, especially dogs. If a dog scratches a couch, its owner becomes angry and may punish the dog (negative reward). If the dog brings a stick, its owner is happy, and will stroke it (positive reward). This way, the dog learns how to maximize its rewards by maximizing the satisfaction of its owner over time. In machine learning terminology, the dog has gradually worked out a *policy* to optimize its performance.

Self-driving cars or optimizing marketing strategies are examples of areas where reinforcement learning can work very well.

2.2 Offline and online learning

Another way to categorize machine learning algorithms is based on the ability to learn incrementally.

In *batch learning*, a system is not able to learn incrementally, and it must be trained using all available training data. In this case, the learning process usually takes a lot of time and resources. When learning is finished, the system is deployed in production, and it doesn't learn any more. If new training data becomes available, the learning process must be started from scratch using both old and new training sets. This is called *offline learning*.

In *online training*, the system can learn incrementally, on the fly. The system receives new training instances either individually, or in small groups called *mini-batches*. Online algorithms can adapt to changes very quickly. What is more, training data can be deleted after training, as it is not needed anymore (in contrast to offline learning).

Online learning can also be used to train the system using huge training sets that wouldn't fit into the machine's memory. The training set is divided into parts. The algorithm loads the first batch of data, performs learning operations, and then loads another batch until all data has been used. This process is called *out-of-core learning*. Since out-of-core learning takes time, it's usually done offline and deployed in production only when it's completed. Therefore, online learning might be a confusing name. This is why sometimes this technique is referred to as *incremental learning*.

In this form of machine learning, it's possible to define how fast does the system learn. The parameter that configures this aspect of an algorithm is called the *learning rate*. If the learning rate is high, the system tends to quickly forget old data. If the learning rate is low, the system will take time to adapt to rapid changes. Optimization of the learning rate in online systems is one of the biggest challenges.

2.3 Instance- and model-based learning

Machine learning algorithms can be categorized depending on their approach to *generalizing*. In other words, how they make predictions. It's not enough to generalize well to examples from the training set. The goal of each machine learning system is to make good predictions to instances it has never seen before.

2.3.1 Instance-based learning

The most straightforward way of learning is learning by heart. If a system remembers the shape of the letter A, it will be able to recognize this letter in all words written using the same font.

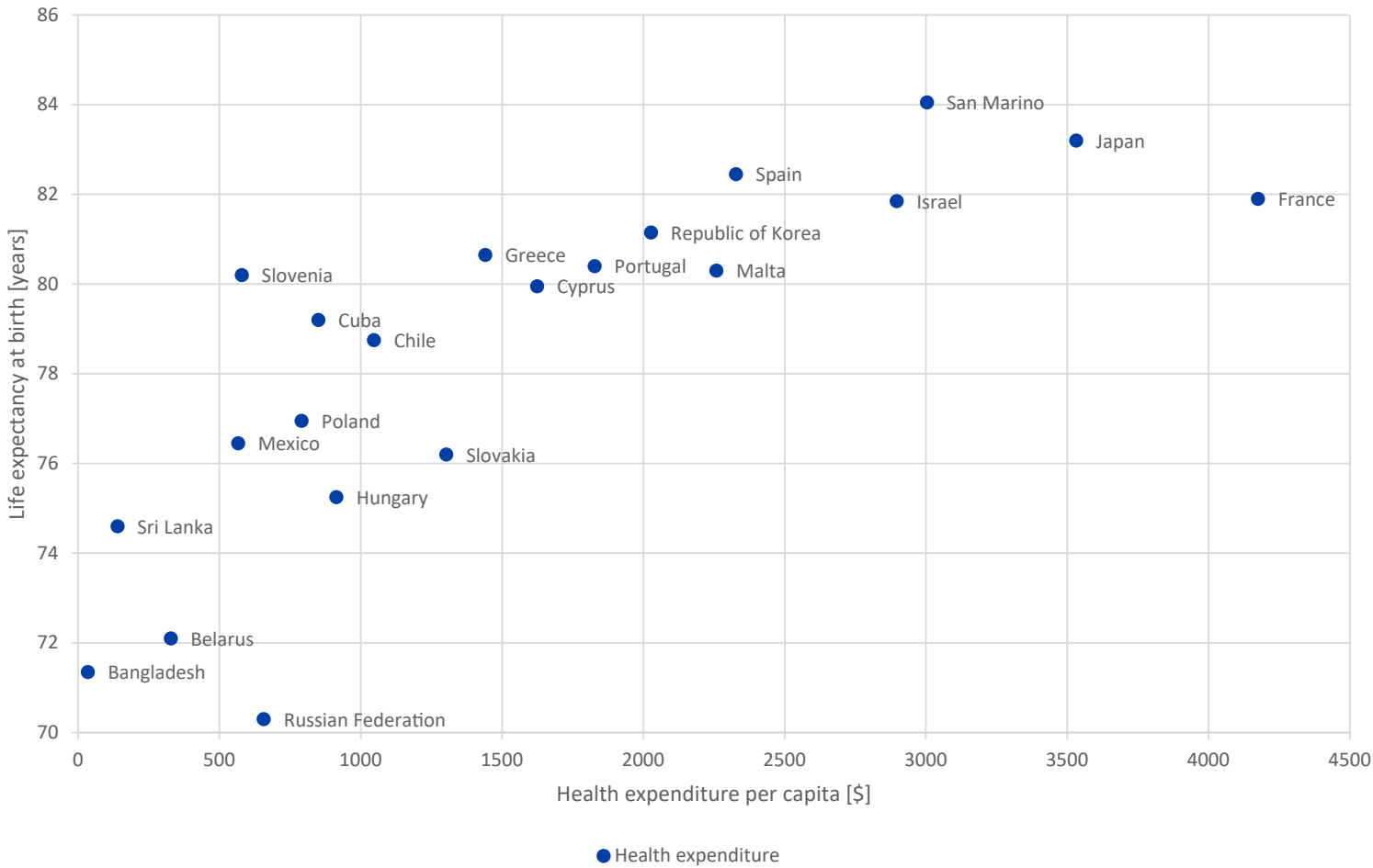
The system can also be programmed to find letters similar to the one it has already learned. In this case, there is a chance to identify the letter A written using other fonts. However, the algorithm needs a *measure of similarity* between the two letters. The measure of similarity is also called *distance function*. An instance-based system compares new instances to the instances from the training set. The closest instance, or instances, are used to predict the output.

2.3.2 Model-based learning

Another approach to generalizing is to build a model during training and then use this model to make predictions.

Let's assume we want to know if rich people live longer than poor people. What we can do is download statistics from the Undata [3] website about various countries' GDP per capita and total health expenditure to learn how much money is spent around the world on health services. Next, we plot this data against the life expectancy at birth for a few random countries.

Figure 1 Health expenditure



Although this data is a bit noisy, it seems that there is a linear trend. The more money is spent on healthcare, the longer the life expectancy is. A linear model has two parameters: a and b.

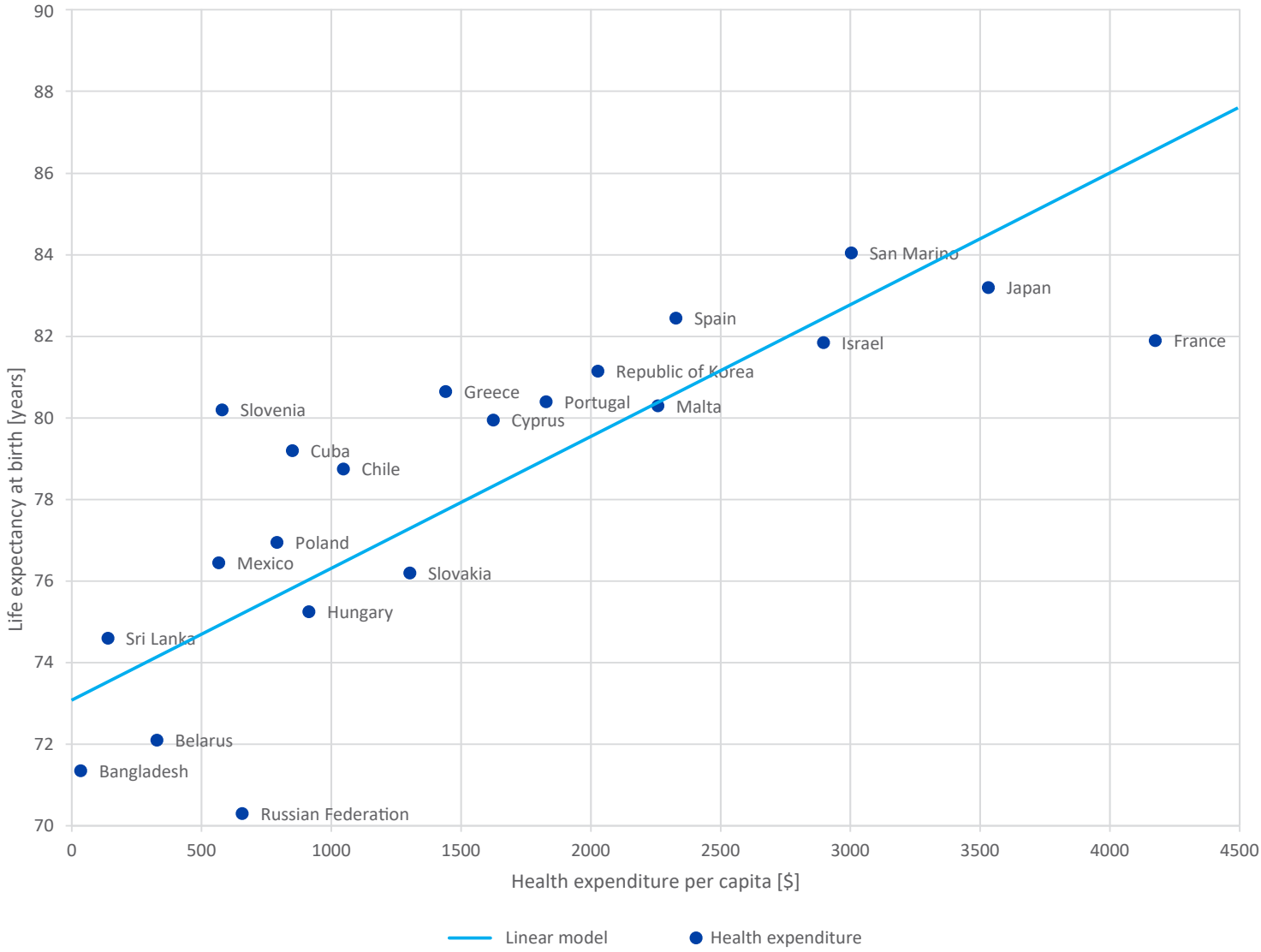
$$y = a \times x + b$$

During the training process, a system must configure a and b to create a performant model for future predictions. To check whether a and b are set optimally, we need to define a *utility function,* also known as the *fitness function*. It measures how good the model is at generalizing. If we define a *cost function* instead, it will measure how bad the model is. In linear regression, the cost function is the average squared distance from training instances to the linear model's prediction. The aim is to minimize this distance.

In our case, during the training, the a parameter was set to 0,0033 and b to 72,96.

Our model is ready to generalize. For example, if there is a country where yearly health expenditure per capita equals to $1500, our model predicts that the average life duration is around 78 years.

Figure 2 Health expenditure



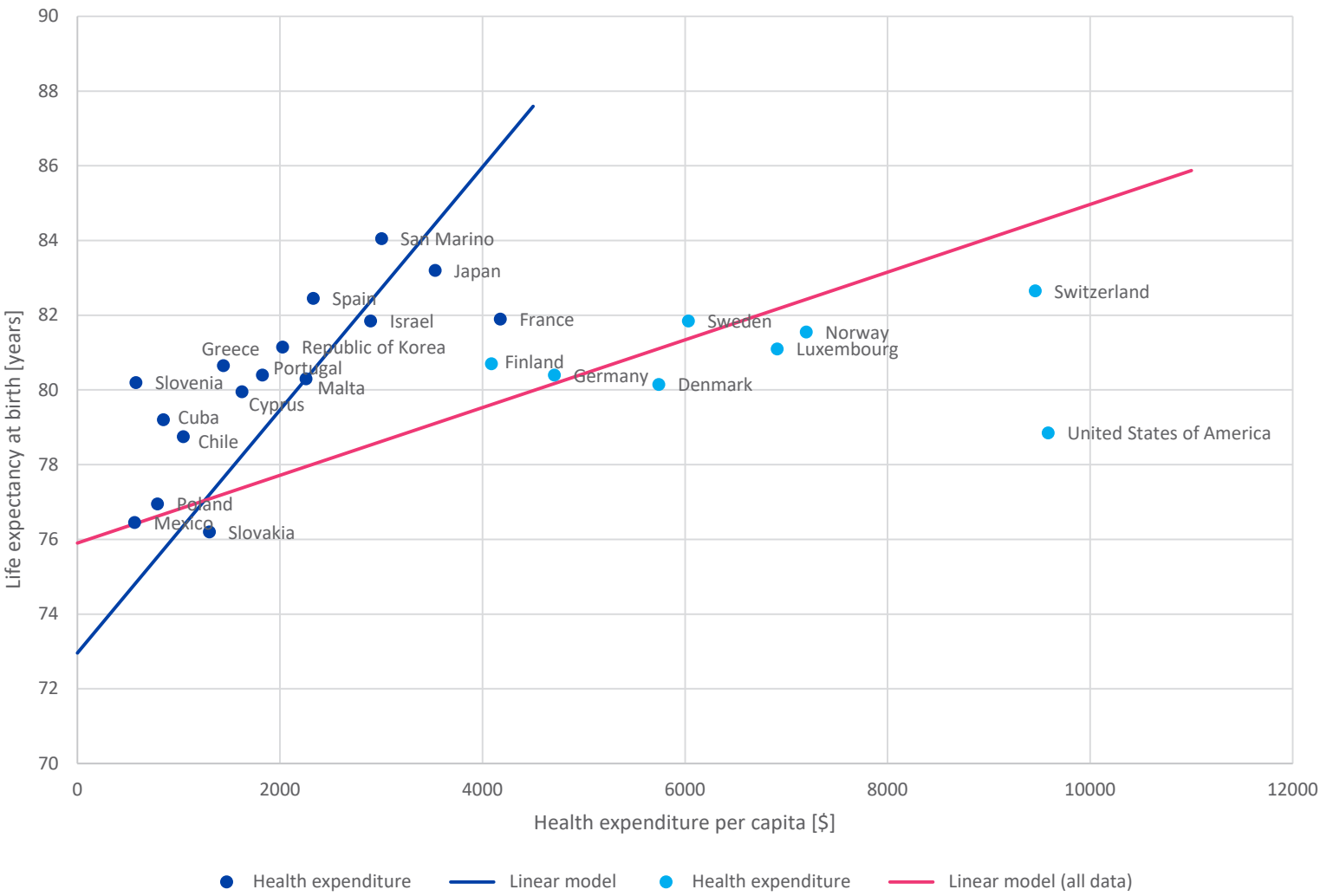
3. Machine learning challenges

When dealing with machine learning, it's important to remember that both the right training data and the right algorithm are equally important.

3.1 Training data

If a training set is small, it might not be representative. In most cases it leads to *sampling noise*. If we add a few more countries to the chart representing the life expectancy at birth (red dots), we will notice that they don't fit well into our model. If we go through the learning process once more, the *a* parameter will be set to 0,0009 and b to 75,90.

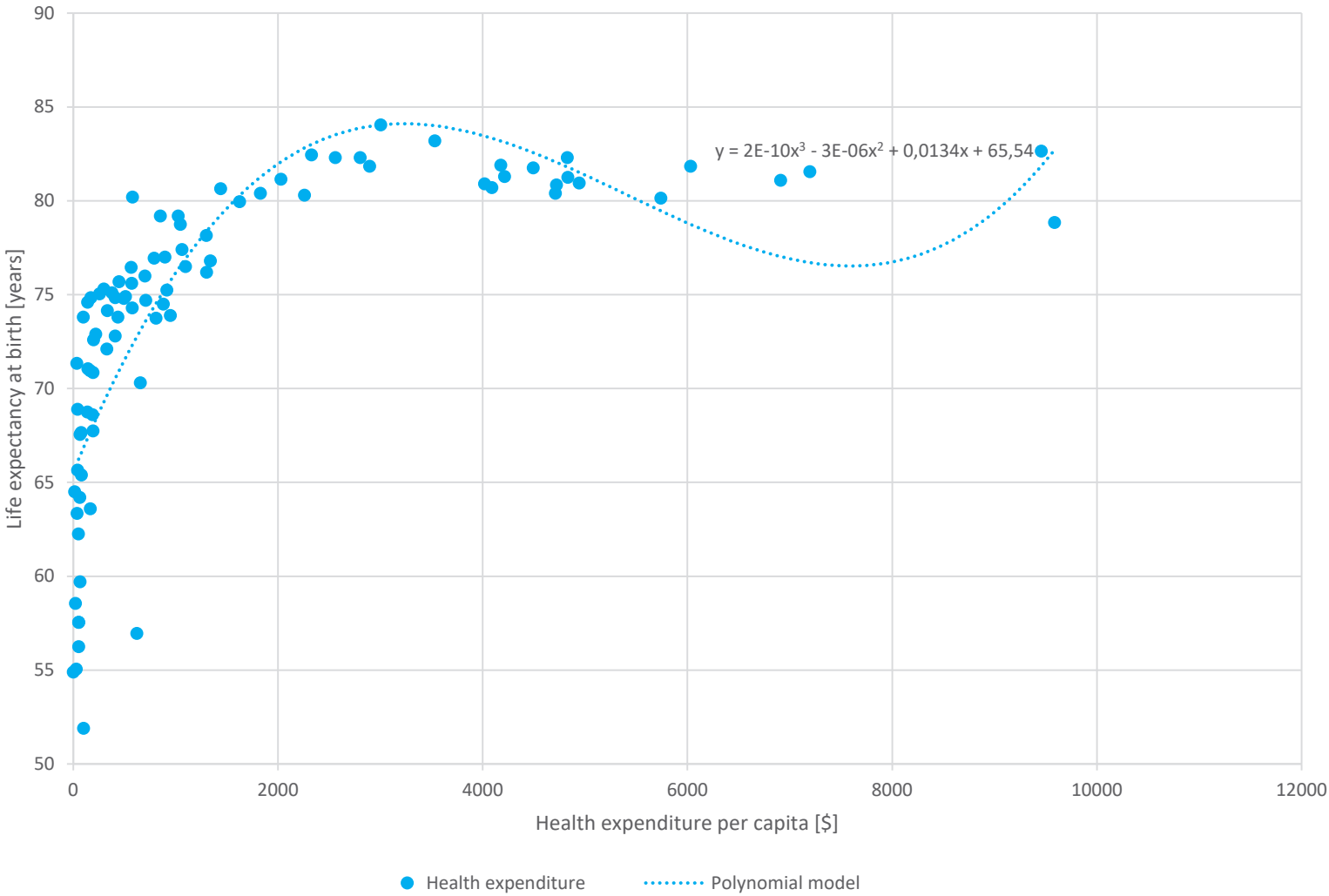
Figure 3 Health expenditure



But even if we collect a lot of training data, it doesn't guarantee that they are representative. For example, we want to know how much money people spend on health care in our country and we make a survey only among our colleagues from work. Even if we collect hundreds of responses, our interviewees don't represent the whole society. This leads to the so-called *sampling bias*.

When we find representative data, it might turn out that the linear model is simply not able to make accurate predictions, as the problem is much more complex. In our case, the average life expectancy depends on many factors like crime level, pollution, or climate. These factors are called *features*. Since a linear model is too simple, we could use a polynomial model instead to improve the generalization performance.

Figure 4 Health expenditure

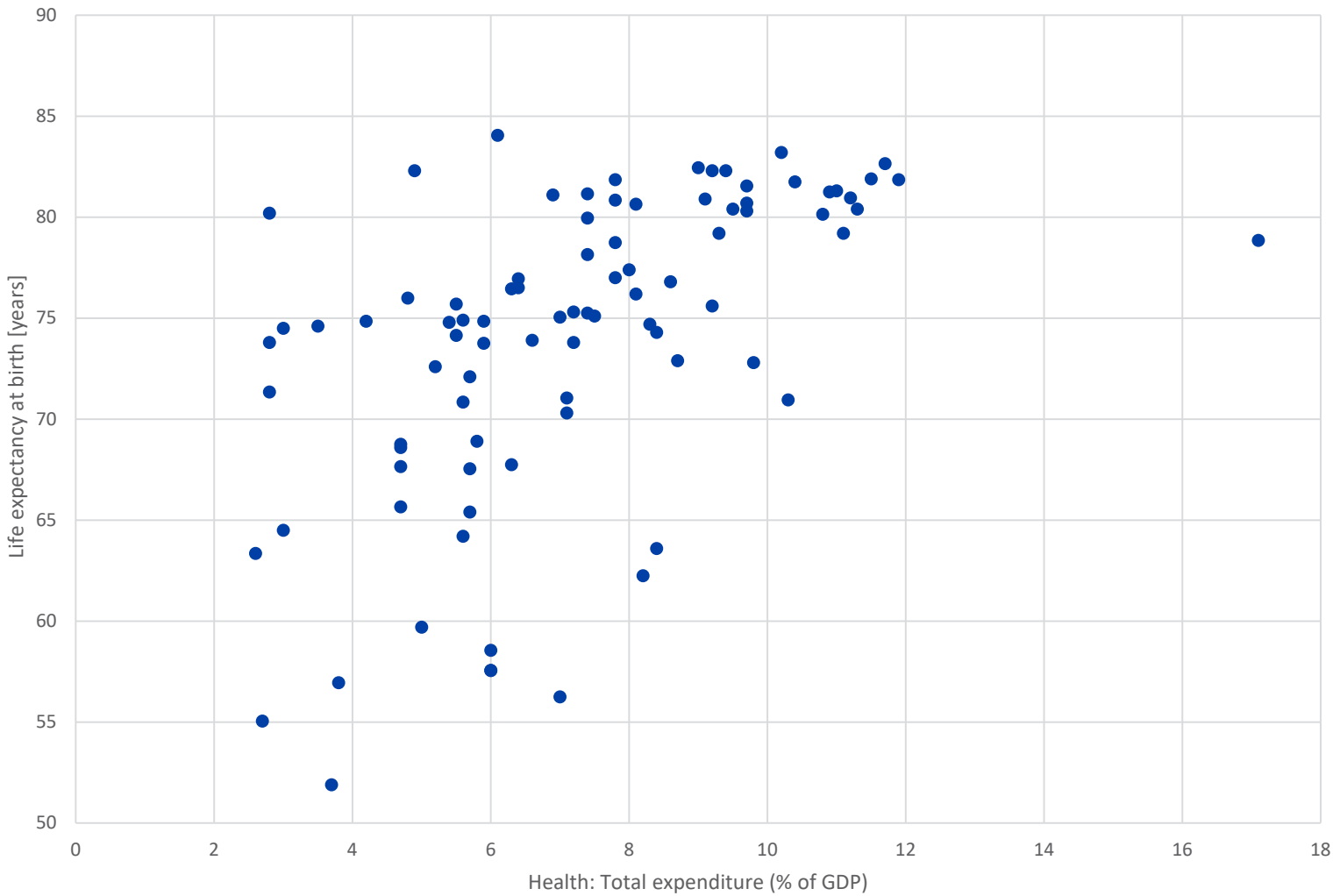


If the training set contains a lot of poor-quality data like outliers, missing or inaccurate features due to measurement issues, the system is unlikely to generalize well.

Another source of problems might be *irrelevant features*. In the chart below, we plot the total expenditure on health as percentage of the GDP. We can see that the result is fairly random, so the machine learning algorithm is not likely to generalize well to new examples.

The key to success in training machine learning systems is to prepare a proper training set. This process is called *feature engineering*. We can combine existing features to create new ones, like in the examples above, in which we calculated the absolute amount of money invested in healthcare.

Figure 5 Health expenditure

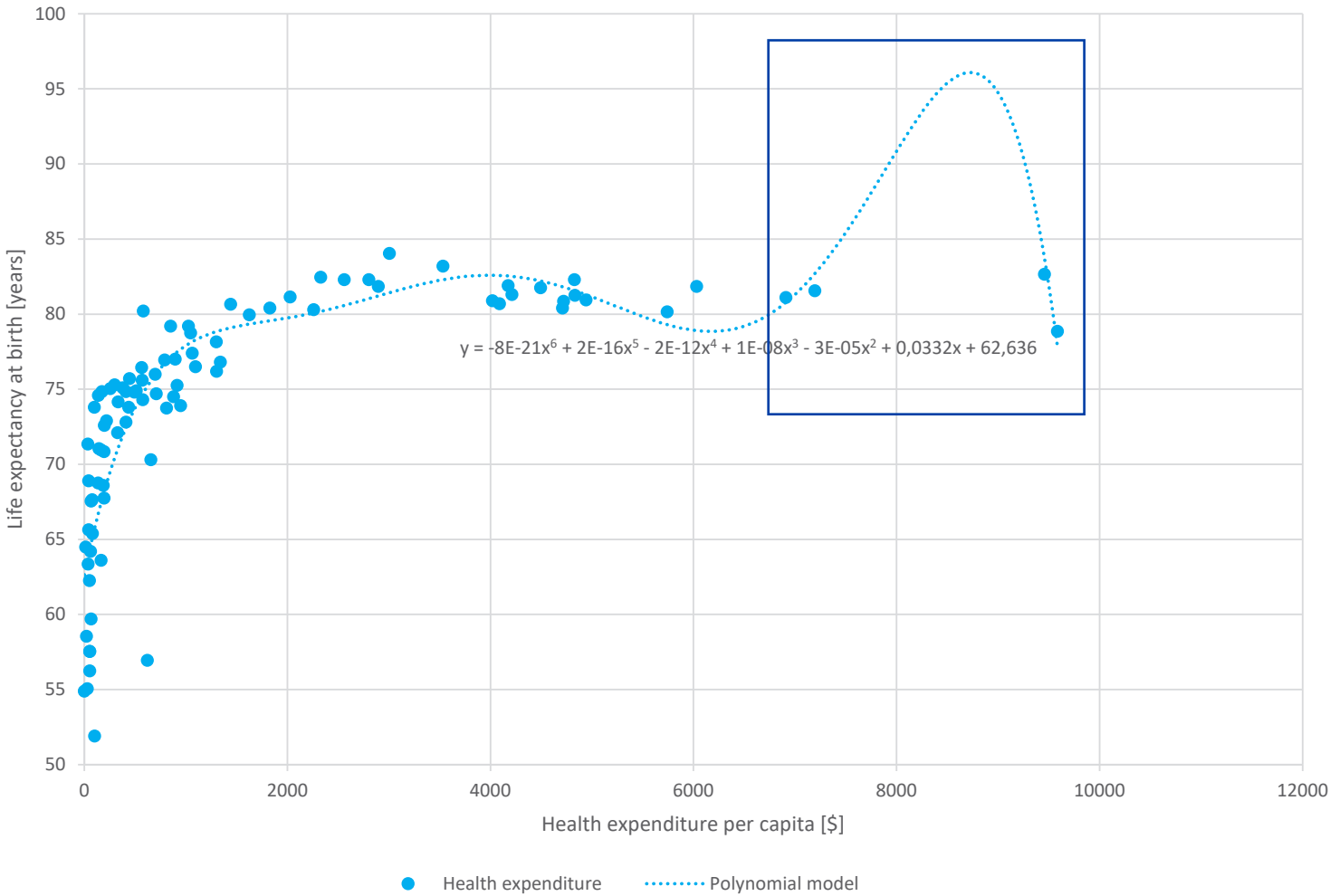


3.2 Bad algorithm

Choosing a good algorithm is another challenge. If the algorithm gives great results on the training data, but its performance drops significantly when doing predictions on new data, it means our model *overfits the training data*. Overfitting occurs not only in machine learning, but in the real world too, all the time. Have you ever heard that every politician is a liar? If someone has seen an example of a politician who was lying and based on this fact assumed that all politicians are the same, he or she is overfitting.

The figure below shows an example of a 6-degree polynomial model that overfits our training set.

Figure 6 Health expenditure



If a model is too complex or flexible, which means not properly regularized or with too many input features, it can learn noise instead of finding the real pattern. Such model will then generalize based on that noise. It will perform well on the training data, but poorly on new examples.

To remedy that, regularization can be used. It's a technique of simplifying complex models. It minimizes the risk that the noise in the training set will impact the predictions on new data.

A linear model has two parameters: a and b. If we allow the algorithm to modify only one of them, we simplify the model. Alternatively, we can allow modification of both parameters, but within a predefined range. This is another example of regularization.

Underfitting is the opposite problem. It occurs when a model is too simple, or too regularized. Underfitting means that the model is inflexible in learning from the dataset.

Apart from regularization, there are also certain *hyperparameters* that must be defined before starting the learning process. A hyperparameter is a variable related to the learning algorithm (not to the model). A few examples of such parameters from different algorithms would be:

- Learning rate
- Number of clusters
- Number of hidden layers in a deep neural network

There are many techniques to optimize hyperparameters. The easiest is just to run the algorithm with a few random values of hyperparameters, to check which combination works best.

3.3 Testing and validating

A common practice is to divide available data into training and testing sets. A testing set can be around 20% of all data, and it can be used only once to verify system performance. The error rate on the testing set is called *generalization error* or *out-of-sample error*.

To define hyperparameters and to select an appropriate model, we need to use the training set. We can extract a subset of data called the *validation set from the training set*. The training set is used to check different models and hyperparameters. Performance of different options is then checked using the validation set. Once the best settings are defined, there is still one more checking stage, in which we use the testing set to estimate the generalization error.

If the training set is small, or if data is expensive, and we don't want to waste some of it for validation, we can use *cross-validation*. The training set is split into k complementary subsets. Each model or combination of hyperparameters is trained using different k-1 subsets and verified against the remaining one. Once the best settings have been defined, the model is trained using the full training set and the out-of-sample error is calculated using the testing set.

4. Summary

This article covered some of the most important terms and concepts in machine learning. It explained basic rules on how to:

- prepare training data.
- select an appropriate algorithm.
- fine-tune and verify the performance of a machine learning system.

While being familiar with machine learning theory is very important, in reality, getting your hands dirty and in experimenting with this new technology is the most exciting. Only then will you discover its unquestionable potential.

References

[1] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2017.
[2] D. Randall Wilson, Tony R. Martinez, *Reduction Techniques for Instance-Based Learning Algorithms*, 2000
[3] http://data.un.org
[4] Krzysztof J. Cios, Witold Pedrycz, Roman W. Swiniarski, Lukasz Andrzej Kurgan, *Data Mining: A Knowledge Discovery Approach*, 2007
[5] Andrew Ng, *Unsupervised Learning*, Stanford University
[6] Jason Brownlee, *Supervised and Unsupervised Machine Learning Algorithms*, 2016

About the author

I studied telecommunication and computer science at the Wroclaw University of Technology, where I focused on the new methods of solving transportation problems. ML and AI seem to be a great extension of metaheuristic algorithms and that's why I find it so interesting.
Today, I lead one of the customer documentation teams in Mobile Networks and I'm responsible for the development and introduction of Discovery Center, the new portal that will be a customer-facing gateway to product information.
I love new technologies that make our life easier. I truly believe that machine learning is the future. My goal is to apply artificial intelligence in customer operations to bring user experience to the next level where even sky won't be the limit.

Adam Kąkol
Squad Group Lead
MN CDS Customer Documentation

Knowledge Engineering

Dominik Dulas
Change Leader
GS CS Network Engineering

Krzysztof Waściński
Product Manager, Machine Learning Guild master
GS CS Network Engineering

NOKIA

The emerging areas of machine learning, and artificial intelligence require knowledge as an input – properly designed, structured, managed and engineered. The better structured it is, the more effective development of AI applications can be. Additionally, creating rich and valuable knowledge sources in a large company like Nokia requires a lot of sharing and collaboration. Multiple knowledge sources promise better quality, but also pose a challenge on how to aggregate information efficiently.

This paper discusses the definition of Knowledge Engineering and the blueprints that Knowledge Creation & Management architecture requires in today's analytics systems.

1. Introduction

One of the most important general-purpose technologies of our era is artificial intelligence (AI), particularly machine learning (ML) — that is, a type of data analysis technology that extracts knowledge from data without being explicitly programmed to do so. Within just the past few years machine learning has become far more effective and more widely available. It is possible now to build systems that learn how to perform tasks on their own.

> *"If you still can't find things within the window of time you need them, you're not doing Knowledge Management right."*
>
> Seth Earley, Founder & CEO, Earley Information Science [1]

Since the most important aspect for AI systems is the right data, first we need to lay the foundation, that is, a knowledge creation and management system. The activities related to building and maintaining such systems are referred as knowledge engineering (KE). Creation, collection, analysis and distribution of knowledge from various human and data sources must be adapted to create an AI-friendly ecosystem.

Knowledge engineering underpins everything we will do in future as we move gradually towards AI. To be successful, KE requires foundational data and knowledge architecture, as well as knowledgeable people with a collaborative mindset. Both these aspects will be elaborated on in the next chapter. The practical examples of Knowledge Engineering will be provided in the final one.

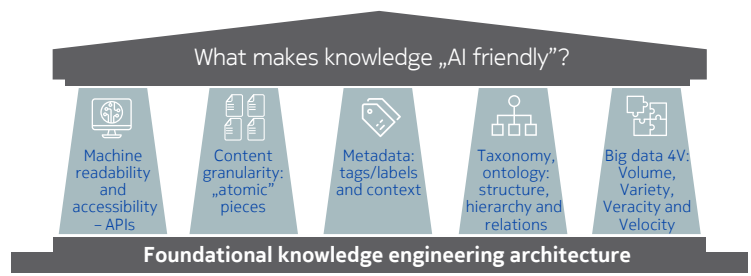
2. Realization

2.1 Foundational knowledge engineering architecture

We have no shortage of knowledge, whether at work or at home. However, the problem may be often the lack of digital access to that knowledge, its lack of clear structure, or other factors that prevent non-human agents from analyzing it and making decisions basing on that. The reason for that problem is that AI tools must be "trained" before they can perform their tasks properly. Thus, all of them require a foundational knowledge engineering architecture. [1]

This is why, before we start actual fun with Machine Learning, we need to enable such architecture. The way we collect data, store information, register decisions and keep all of them must be adapted to create an ecosystem which will be "AI friendly". **Figure 1** presents the main elements of that architecture, which we will explain in detail in this chapter.

Figure 1 The foundations of Knowledge Engineering



2.1.1 Machine Readability

The first step is to make the data easily readable for a computer and ML scripts. That means that inaccessible sources like emails, local files or simply knowledge in your experts' minds need to be migrated out to some database, so that they can be indexed centrally and made searchable. Obviously, non-digital material (for example printed or hand-written documents) is by its non-digital nature not machine-readable.

Secondly, once data stores are established they could be extended with APIs (Application Programming Interfaces). This would allow also knowledge exchange without the need of downloading the whole dataset, thus saving bandwidth and ensuring that the used data is the up-to-date.

Bad examples: formats which needs a lot of computation to obtain the information stored there, i.e. PDF document containing tables of data; whiteboard photos in the mobile phone, emails discussion in your mailbox, a paper book.

Good examples: Structured data in Cloud accessible databases, files in the O365 Cloud (i.e. SharePoint), XML, JSON, CSV files.

2.1.2 Content granularity

For many ML use cases very detailed data (atomic pieces) is required to make use of it. For that reason, knowledge needs to be collected in standalone, reusable, format-independent pieces. Content granularity refers to the size of those data fields; in short: how detail-oriented a single field is. Please note we need to keep it balanced, as finer granularity has overhead for data input and storage.

Bad example: just plain text document, name or address field being saved as a whole.

Good example: document divided into small chapters, name, address, city, postal code being saved separately in multiple fields.

2.1.3 Metadata

Every piece of data has a specific context under which it is valid. Therefore, we often need additional metadata – which is simply information about another item's content or its other aspects, which can make tracking and manipulating it easier.

Bad example: just plain raw data without the contextual information.

Good example: data tagged with additional contextual information, e.g. customer name, date or period, source, tester, purpose, collection procedure.

2.1.4 Taxonomies

One of the standard techniques to build well-structured knowledge bases is to provide meaningful categories and add semantic relationships between the data pieces to explain their meaning. Those techniques are called taxonomies and ontologies.

A taxonomy is a simple hierarchical arrangement of entities where you have a parent-child relationship. Imagine an online-store where products are categorized at many levels to make it easier to search for the thing you need, e.g. *Electronics – TVs – Antennas – External antennas*. It is also very similar to the taxonomies that you study in the field of Biology, e.g.: a *Panther* is a type of *Carnivorous animal*, which in turn is a type of *Mammal,* and all Mammals are types of *Animals*. Taxonomies are simple arrangements of classes without any restrictions on any properties at any level of the hierarchy.

An ontology, on the other hand, is a more complex variation of taxonomy. Besides having the hierarchical arrangement of the classes that represent entities, each class has several internal restrictions on its relationships to other classes or on the properties a particular class is allowed to possess. Differences between these subjects can be very controversial in some academic circles [2].

Bad example: just raw eNodeB Key Performance Indicator data.

Good example: data which is described in reference to other data: counters within the KPI, cell-to-BTS ID, network area, customer and country mapping, etc.

2.1.5 Big Data aspects

"Big Data" as a set of hardware and software solutions used to capture, store and process large amounts of data to produce a meaningful outcome. Some people claim that it is just a buzzword,. The truth is, while they may not be directly visible to end users, those technologies fuel now every major service of companies like Google, Netflix or Amazon, as well as at Nokia. Hadoop, Cassandra, HBase database systems and Spark distributed query engines, Business Intelligence tools, PowerBI and many others – those are now a commodity, not buzzwords. Big data 4V elements (Volume, Variety, Veracity and Velocity) are essential for AI engines to provide enough algorithm accuracy and Actionable Intelligence.

Bad example: small chunks of BTS logs with overlapping periods, missing data and various files format, since they came from different monitoring systems

Good example: Cleaned, filtered and merged dataset of relevant log data points, large enough to train the neural network (please note that size depends on the problem being tackled)

2.2 Power of Collaboration

> *"It is amazing what you can accomplish if you do not care who gets the credit."*
>
> Harry Truman

A definition of collaboration in the Knowledge Management (KM) context: "*Effective method of transferring 'know how' among individuals, therefore critical to creating and sustaining a competitive advantage. Collaboration is a key tenet of KM.*" [3]

In the Internet era we have access to humongous amounts of data, information and knowledge, which is a wealth but also a burden. On the one hand, in theory we can find any kind of information that is stored somewhere in the net. However, if you want an insightful answer to a specialized technical query, it is not really that easy as one would think. However, we believe that this is about to change with the rise of a new set of augmented intelligence tools that enhance the human thought process. At first, these tools will be able to search through the ocean of data and locate the valuable information. Later, thanks to machine learning or deterministic algorithms, they will be able to support humans in their thought process by giving the right answers, e.g. via a chatbot interface. In order to feed this new set of augmented intelligence tools, we first need to create a way of storing knowledge, once available to individuals but also to machines. Concepts described in next chapter will describe how to do that.

2.2.1 Data democracy

To fully grasp the power of data, first the data needs to be available to everyone looking. Assuming that you have a well formulated problem, to be able to find anything first you need to know where to look. Next, you need to have access to the data. Finally, you need to know how to digest it to get answers you are looking for. To achieve all of this, an organization needs to implement several strategies, both from process and culture sides.

The processes must guarantee that when a specific data source is discovered there is a clear way how to make it digital, accessible and visible for any audience. It should not require any lengthy business justification and second line managers approval.

Company needs to create a culture of openness by establishing data democratic practices and processes. .

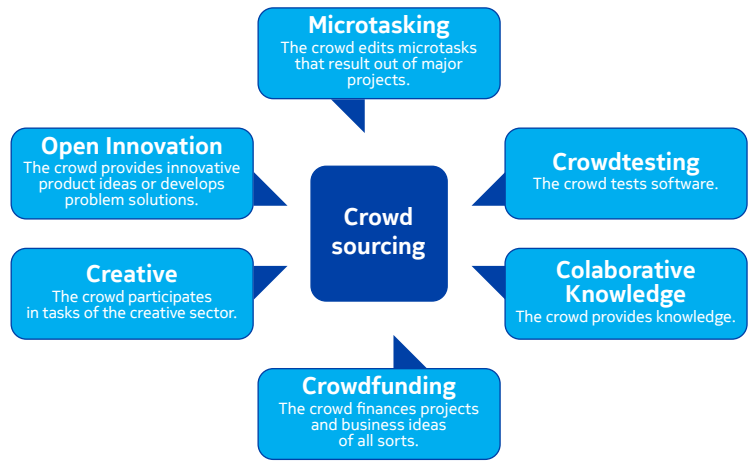
Finally, democratized data should be understandable for non-specialists. This is probably the hardest nut to crack, since it's hard to define what is the level of abstraction, generalization or contextualization that is understandable for non-specialists.

2.2.2 Wisdom of the crowd and crowdsourcing

Wisdom of the crowd (WotC) is the idea that groups of people are collectively smarter than individual experts when it comes to problem solving, decision making, innovating and predicting. For example, when averaging the individual guesses of a large group about the weight of an object, the answer may be more accurate than the guesses of experts most familiar with that object.

Crowdsourcing is the act of outsourcing a job usually done by a single person to a large group of people. The goal is to distribute workload from one to many individuals. For example, tagging pictures with the objects that are in them by a large group of people, testing a feature, submitting a new idea or gathering the funds (**Figure 2**). At Nokia, we apply both these concepts. We use crowdsourcing to be faster and more effective in basic tasks, e.g. by collecting mobile data in scale with a shared Android application instead of expensive drive testing (that is described in one of the examples in next chapter). On top of that, we harness the engineering know-how, e.g. on our product parameterization, of our experts to prevent any biased opinions and enable faster knowledge creation and better information flow.

Figure 2 Crowdsourcing examples



2.2.3 Scaling knowledge

Artificial Intelligence tools are not yet ready to generate insights as well as humans do, and there is a big dispute whether they will ever be able to. Because of that, efficient ways to transfer 'know how' between individuals must be created.

When your team consists of several people it's quite easy to share your discoveries, but what about organizations that have hundreds of teams, how would you know how to inform the right people? In other words, how to make knowledge scalable?

There are two things required to scale knowledge, a process and a tool. The process is needed to guide individuals how to be a part of a bigger team, formalize and save the know-how. It needs to describe how to contribute and review the work. You can't scale crappy code – the same ruls applies to knowledge. You need to take

care of the quality of the knowledge from the smallest insight, and set a way to verify it at every stage. “Unlike engineering code, low quality research doesn’t create metric drops or crash reports. Instead, low quality research manifests as an environment of knowledge cacophony, where teams only read and trust research that they themselves created.” [4]

In combination with the process, an accessible storage is required. It needs to be well organized to showcase the know-how for both people ands machines. Accessibility means not only a graphical user interface but also a way to distribute information, locate and notify the potentially interested recipients. At Nokia, we build tools and databases customized to suit our engineering processes.

3. Examples

3.1 Use-case: Knowledge brokering

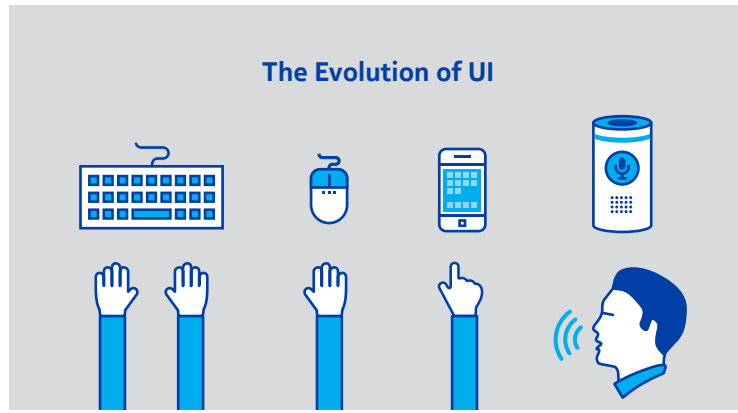
Handling network data is inherent part of Knowledge Engineering within Nokia. That means that, we know precisely how Nokia products are deployed in the field, and what is the exact configuration and performance of customer networks served by Nokia. This process of exposing the network data so that it could be quickly consumed by diverse organizations is what we call smart data brokering. Intelligently processed and exposed network data fosters R&D, operational speed and effectiveness. Smart network data brokering empowers business management operations, optimum product design, lean testing, fast product introduction and other network operations.

The exposal of smart network data is done in two ways, via APIs to connect it with intelligence systems which can automate or augment engineering and business processes, and via graphical dashboards which provide authoritative, reliable and holistic view of the customer network– starting from precise and up-to-date information on setup of field equipment down to a detailed overview of network performance, customer benchmarking figures and identified capacity bottlenecks. These dashboards convey wisdom extracted out of network data to support all customer-impacting decisions and to trigger relevant actions right in time – with embedded outside-in approach.

3.2 Use-case: chatbots

The Success of commercial chatbots and assistants results from the conversational interface, which simplifies human-computer interaction. It is a paradigm shift from the earlier communications achieved either by entering syntax-specific commands or clicking icons. Thanks to the wide availability of the devices with in-built microphones, it evolved into a conversational user interface that allows a user to tell the computer what to do (**Figure 3**). In the past chatbots were simply rule-based (if the user says A, the bot will say B). Further Natural Language Processing (NLP) advancements helped to analyze the speech data, but still a chatbot was not looking at its meaning, therefore it was unable to detect the intent of the user. The invention and refinement of deep learning technologies such as text-to-speech, automatic speech recognition and Natural Language Understanding (NLU, enabled chatbots to track the conversation context and recognize ontologies. That simply would not be possible without dedicated, ML-based, neural network architectures.

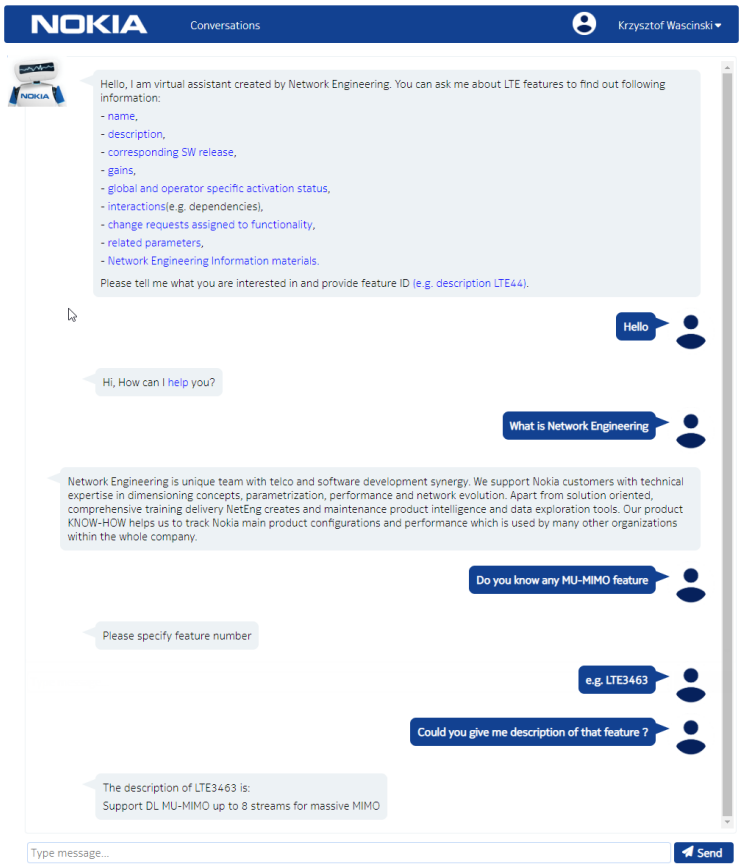
Figure 3 Evolution of UI



Today chatbots that simulate human dialogue can be found in first-line customer support, in information acquisition and search systems. They can provide highly engaging, conversational interaction, between the user and system, through voice and text. The experience can be customized and via virtual assistants (Google Assistant, Amazon Alexa), messaging apps (Facebook Messenger, Slack, MS Teams) and websites.

One example of such a chatbot in Nokia is the *Network Engineering Virtual Assistant* (**Figure 4**). Its goal is to ease the access to knowledge about Nokia product features stored in multiple sources. Thanks to that engineers can extract the required information, like feature dependencies, benefits, related parameters or corresponding SW release information, about two times faster than, by going into each of the tools separately. It would not be possible to create this chatbot without applying knowledge engineering principles to all of its knowledge sources.

Figure 4 Network Engineering chatbot prototype



3.3 Use-case: telecommunication data crowdsourcing

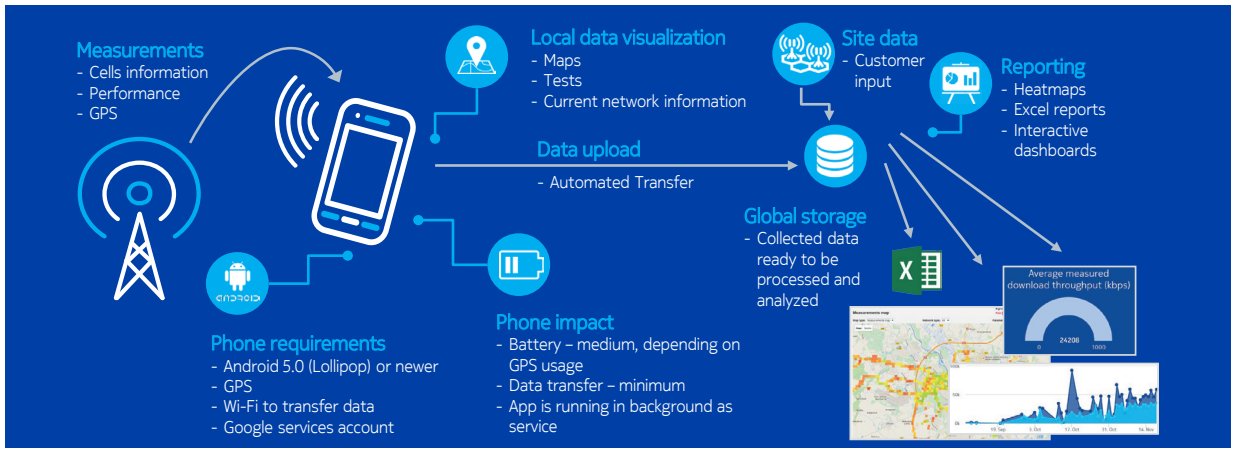
One of the exemplary applications of the crowdsourcing concept is smartphone-based network monitoring. Such apps can provide valuable insights on the mobile experience, like throughput or latency perceived from the user's perspective. Compared to common eNB metrics, they're not averaged over the longer time periods and groups of UEs, so they can provide finer granularity. Collected from thousands or even millions of distributed mobile devices, this data can help operators to:

- accurately benchmark network performance, optimize their networks and service quality,
- enable improved consumer decision making,
- automate network rollout activities.

In our Network Engineering department, we ran a crowdsourcing project in close cooperation with one of Polish mobile network operator, to deliver a *PROBO* solution, consisting of an Android app, a database and a reporting platform. The initial version of the app was created during an internal hackathon, and then actively developed towards becoming an official Nokia service, offered to external customers. Figure 2 presents an overview of its architecture and the flow of the data through the system.

Once data was collected in the global storage, it was exposed to a Business Intelligence (BI) application to create immersive reports with various types of charts or maps. But the real value of that data can be unleashed later by joining it with others, like the performance metrics measured by the network equipment. Tha is why exposing it further through the API is even more important.

Figure 5 PROBO solution architecture



In any kind of crowdsourcing applications, the main challenge is to convince the end users to install the app and allow it to take the measurements. In our case, the application was simply distributed among the Nokia employees who volunteered to conduct the measurements on their way to the office or during travels. In general, the most popular app currently available on the market channels the users' need to measure network speed (to e.g. verify if it meets the values promised in their mobile plans), using this to motivate them to download the app. Another approach would be a "silent" installation of measurement clients as a part of some other application (i.e. a mobile game), so the user doesn't know if their mobile is sending the data (unless they've read all the app's permissions). The success of such apps relies on the scale of the measurements taken, so performance evaluation is statistically reliable.

4. Summary

In reality, we have no shortage of knowledge. But some way of collecting data, storing information, registering decisions and keeping information about them must be adapted if we want machine learning to happen. Thus, Knowledge Engineering is more important now than ever. It is a continuous process, whose results will come with time. Content is key, as new technologies (AI, ML Chatbots) do not operate in vacuum.

How can I contribute, then?

Create with structures
- When you create documents or reports, think about structuring them for machine readability
- Generate content automatically, not by hand
- Store data in some database instead of Word, Excel or PowerPoint documents.
- Build tools and databases customized to your engineering processes.

Share
- Support data democracy culture so that sharing of data sources is in the bloodstream of your colleagues
- Share that knowledge/data by storing it in cloud/open tools, to make it harvestable by ML
- Contribute to crowdsourcing of data by allowing your data to be collected by 3rd party components.

Enrich
- Every time you're sharing any of those data– keep in mind to provide metadata and labels – if possible, by matching those used in the past, to give it meaning and a relationship to the wider ecosystem.
- The multiplicity of systems and their interfaces around requires a high dose of engagement and learning. It also requires that there be some way to discover the relevant system
- Once content becomes available, machine learnin, and AI will help to discover hidden patterns in that data, and will yield highly useful information that can in turn improve human efficiency.

References

[1] "Earley Information Science," 2018. [Online]. Available: http://www.earley.com/topic/knowledge-management.
[2] "New Idea Engineering," February 2018. [Online]. Available: http://www.ideaeng.com/taxonomies-ontologies-0602.
[3] [Online]. Available: http://www.businessdictionary.com/definition/collaboration.html.
[4] C. S. a. J. Overgoor, "Scaling Knowledge at Airbnb," [Online]. Available: https://medium.com/airbnb-engineering/scaling-knowledge-at-airbnb-875d73eff091.

About the authors

Leader of transformational changes improving the way of working across whole Network Engineering organization. Over a decade of experience in telecommunication engineering across all domains, radio, transport, core and cloud. Supporting and promoting innovative projects, especially related to Knowledge Engineering and Machine Learning.

Dominik Dulas
Change Leader
GS CS Network Engineering

Tribe Product Manager responsible for managing backlog, prioritizing and guiding the work of the Network Engineering organization. Experienced telecommunication engineer with LTE and 5G background. Leading the execution of iconic customer projects in area of system-level simulations and network performance analytics. TC-Wroclaw Machine Learning Guild host and AI enthusiast.

Krzysztof Waściński
Product Manager, Machine Learning Guild master
GS CS Network Engineering

AdaBoost algorithm for data exploration

Maciej Kondrat
Data Scientist
NOPS CMBD CSBD AI Lab

Paweł Skoliński
Data Scientist
NOPS CMBD CSBD AI Lab

Witold Pawlus
AI Lab Lead
NOPS CMBD CSBD AI Lab

NOKIA

Modern machine learning techniques are often based on the concept of using multiple classifiers to achieve better accuracy. "Boosting" is a methodology that can significantly improve accuracy of a weak learner by aggregating the results of refitting this algorithm to a dataset multiple times. Enhancement of the algorithm's performance is achieved by using the error from the previous iteration to improve ("boost") the results in the next iteration. In this article we present the AdaBoost algorithm, the first practical boosting implementation, known for its ease of tuning and robustness to noise in data.

1. Algorithm Description

AdaBoost (abbreviation of Adaptive Boosting) is one of the first approaches to boosting methods in machine learning. Proposed by Freund and Schapire in 1997, the algorithm consists in constructing one strong classifier by combining multiple 'weak' classifiers (often referred to as base learners). This approach can be beneficial for reducing overfitting, thus increasing algorithm generalization.

1.1 Basic idea

The basic boosting idea can be expressed in the following formula

$$H_M(x) = \sum_{m=1}^{M} \alpha_m h_m(x),$$

where $H_M(x)$ denotes a strong classifier, which is the sum over M weak classifiers $h_m(x)$ trained on input data x. In other words, a strong classifier is a linear combination of weak classifiers where each base learner contribution to the final classifier is weighted by a factor α_m.

Denote by N and y_i respectively a number of samples in the dataset and a ground truth label for i-th sample.

The process of learning is iterative and usually repeated a chosen number of times M. At each iteration step m, the algorithm chooses the weak learner h_m that minimizes

$$\varepsilon_m = \sum_{i=1}^{N} w_{i,m} \mathbb{1}(h_m(x_i) \neq y_i),$$

where $w_{i,m}$ is the weight of sample i at iteration m Initially, these weights are set uniformly as $\frac{1}{N}$.

Each classifier's contribution is then weighted with α_m, calculated as

$$\alpha_m = \frac{1}{2} \ln\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right).$$

One of the most important concepts in AdaBoost training is its adaptability. This is achieved by weighting samples according to the current classifier error. At each iteration samples' weights are updated according to the formula

$$w_{i,m} = \frac{w_{i,m-1}\, e^{\alpha_m\left(-y_i \cdot h_{m-1}(x_i)\right)}}{\sum_{i=1}^{N} w_{i,m-1}}.$$

Weights are normalized so they can be considered a valid probability distribution.

This procedure leads to creation of the strong classifier that uses predictions of weak classifiers to diminish the total error, in this case stated as the sum of exponential losses on each data point x_i

$$E = \sum_{i=1}^{N} e^{-y_i H_M(x_i)}.$$

This classifier can be then used at inference stage.

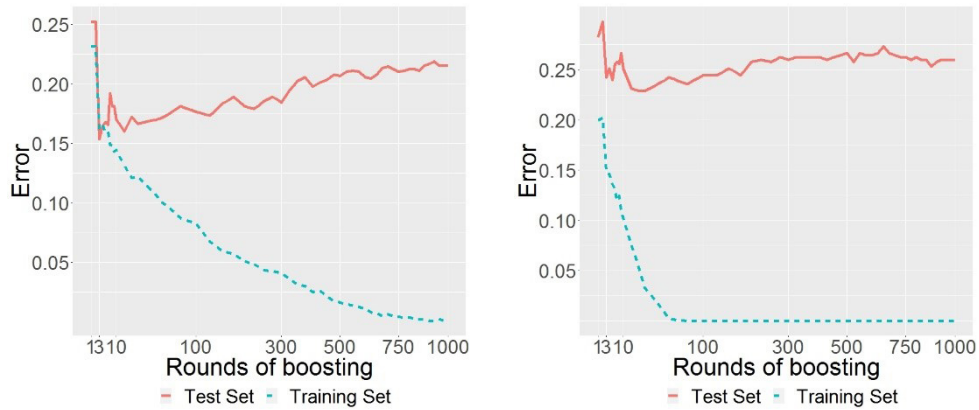
One should note that the described algorithm works in a discrete framework, where error is calculated as a sum of weights for samples that are misclassified. It turns out that the probabilistic formulation, introduced by Schapire (Freund and Schapire, 1997), called SAMME.R gives better results in most classification problems. In the latter approach classifiers output probability estimates for all the classes instead of {-1, +1} labels. The full formulation of this approach is out of scope of this article, but a curious reader is encouraged to take further look at (Freund and Schapire, 1997).

1.2 Weak learners

An important assumption of the learning process is that each of the base learners itself does not achieve good accuracy on the given dataset. It is their combination that leads to significant error minimization as each subsequent learner learns features omitted by the previous ones. Therefore, to prevent overfitting of single classifiers, it is important to keep them as simple as possible. Usually, the first choice for a weak learner is a decision tree with relatively low depth (sometimes even one level). In that case, C4.5 trees are most widely used. Very randomized models could also bring significant benefit, e.g. by using Extremely Randomized Trees (Geurts, 2006). On the other hand, even though neural networks or SVMs are less widespread and successful, nevertheless there exist some promising results regarding these approaches (Li, 2005). There are also examples of using complex bagging models such as Random Trees that perform well on certain classification tasks (Leshem, 2005).

To further reduce overfitting, it is also possible to use an early stopping technique that can halt learning if the error on validation dataset coming from adding subsequent classifier does not change significantly.

Figure 1 The training (blue) and test (red) error obtained using AdaBoost heart-disease dataset; left-hand side: using decision stamps, right-hand side: using trees with maximum depth of two.



1.3 Real-world applications

Variants of AdaBoost have been extensively used in many real-world applications, achieving satisfactory results (Maclin, 1997). These tasks also include computer vision. One of the most prominent applications of AdaBoost before the rise of deep neural networks was face detection. This method, combined with Haar feature extractor (Papageorgiu, 1998) achieved state-of-the-art results in the field. The algorithm was used on numerical features extracted from images and was the first algorithm that could detect faces in both fast and robust way (Viola, 2001). Many modifications of the approach, mainly considering more complex feature detectors, were proposed later on (Lienhart, 2002). These methods were widely used in image processing applications for many years, e.g. for basketball player detection (Mahmood, 2012). Until the emergence of learned image features approaches, AdaBoost was a preferred method for image classification and object detection. It was also used in image retrieval tasks (Tieu, 2000) as well as in OCR, combined with artificial neural networks (Schwenk, 1997).

Other significant applications of this algorithm include text categorization (Schapire, 2000), speaker detection (Zhang, 2008) and medical usages, including breast cancer survivability prediction (Thongkam, 2008).

2. Real-world data example

As mentioned before, decision trees are a typical choice for weak learners for the AdaBoost algorithm. On the left-hand side of **Figure 1** we present results of fitting AdaBoost algorithm with decision stamps (trees of maximum depth one – only root node) to a heart disease dataset (Dua and Karra Taniskidou, 2017), whereas on the right-hand side the same fitting operation is done but with trees of maximum depth two.

Note that for this dataset AdaBoost performed like a classical machine learning algorithm – the error on the training set decreased with every iteration, while at some point the error on the test set started to increase. In other words, the model tends to overfit.

As it can be observed on error curves (error is the fraction of incorrectly classified observations), the algorithm with simpler weak learners has better performance. This result is in line with conventional wisdom that for good classifications, AdaBoost needs to be built using weak learners with low complexity. In the next section we will show, however, that this assumption does not hold for every dataset.

Simulations in this or next sections of this article were made in R environment. We used our own implementation of AdaBoost, presented below. All results were obtained using a 5-fold cross-validation methodology.

Listing 1 Example of AdaBoost with tree-based weak learners implementation in R environment.

```
library(rpart)
library(magrittr)
AdaBoost <- function (X,Y,X_test,
                      n_rounds,tree_control){
  n = dim(X)[1] #number of records
  w = rep(1/n, n) #initial weights for training set
  trees = list()
  a = list()
  for (i in seq(n_rounds)) {
    #adding next boot-iteration tree
    trees[[i]] <- rpart(Y~.,
                        data=data.frame(X),
                        weights = w,
                        control = tree_control,
                        method = "class",
                        x=FALSE,y=FALSE,model=FALSE)
    pred <- trees[[i]] %>%
      predict(data.frame(X),type = "class") %>%
      as.character() %>% as.integer()
    #counting error of prediction for i-th tree
    error = sum(w * (pred != Y))
    #counting weight of i-th tree
    a[[i]] <- ((1-error)/error) %>% log() %>%   multiply_by(1/2)
    #updating weights of traing set
    w <- w * exp(-a[[i]]*pred*Y)
    #normalization of new weights
    w <- w/sum(w)
  }
  Y_pred <- sapply(seq(1,n_rounds),
                   function(i) predict(trees[[i]],data.frame(X),
                   type = "class") %>%
       as.character() %>% as.numeric() %>%
       multiply_by(a[[i]])) %>% rowSums() %>% sign()
  Y_pred_test <- sapply(seq(1,n_rounds),
                        function(i) predict(trees[[i]],data.
                        frame(X_test),type = "class") %>%
    as.character() %>% as.numeric() %>%
    multiply_by(a[[i]])) %>% rowSums() %>% sign()
  return(list(a = unlist(a), trees = trees,
              Y_pred = Y_pred,
              Y_pred_test = Y_pred_test))
}
```

3. Margin maximization

Let us consider a different dataset - the phoneme database available online (Phoneme, 1993). This is a speech recognition task, and the task of the algorithm is to classify vowels into one of the categories: "nasal" or "oral".

Errors curves presented on both graphs of **Figure 2** show two counterintuitive behaviors of AdaBoost. Firstly, the test error does not increase even after 1000 boosting rounds. This is even more surprising if we consider that each new weak learner is trained on a more specialized subsample of data. Secondly, the training error drops to zero after just fifteen trees have been build, but the test error continues to decrease.

According to the work of Schapire et al. (1998), to explain the nature of this seeming paradox we need to consider more than just the training error, but also take into account how confident are the predictions made by the algorithm. As we will see in this example, although the accuracy on the training set for AdaBoost with trees of depth 8 is not changing after 15th iteration, the confidence in those predictions increases with additional rounds of boosting.

Let us define a quantity called *margin*. Recall that the AdaBoost prediction is simply a weighted majority vote of the predictions of the weak learners. The classification margin is a difference between the sum of weights predicting the correct label and the sum of weights predicting the incorrect label, divided by the sum of all weights. Margin of *i*-th observation can be formulated by using the following equation

$$m(x_i) = \frac{|\sum_{m=1}^{M} \alpha_m \mathbb{1}_{R^+}(h(x_i)y_i)| - |\sum_{m=1}^{T} \alpha_m \mathbb{1}_{R^-}(h(x_i)y_i)|}{\sum_{m=1}^{M} \alpha_m}.$$

Note that margin values lay between -1 and 1. A positive margin indicates that the prediction was correct.

On the left-hand side of **Figure 3** we present the cumulative distribution of margin after 15 and 1000 iterations. While the algorithm could not perform better in terms of accuracy error, its confidence has changed. About a quarter of observations after 15 rounds was below 0.35 margin value, which is a minimal value after 1000 rounds. Also note that higher accuracy of weak learners corresponds to larger margins. However, according to the algorithm definition, a weak learner with accuracy equal to 1 will stop the learning process.

It is worth mentioning that margin maximization only captures some aspects of AdaBoost's behavior. The other explanation based on loss minimization or regularization can be find in (Schapire 2013).

Summarizing, AdaBoost is an effective machine learning algorithm, which, if provided with enough data to compare the complexity of weak learners, can learn without overfitting. However, like many other approaches in machine learning, this analysis does not ensure any sufficient condition defining for which cases overfitting is not observed.

Figure 2 Training (blue) and test (red) error obtained with AdaBoost on a phoneme dataset; left-hand side used trees with maximum depth of six whereas right-hand side trees with maximum depth of eight.

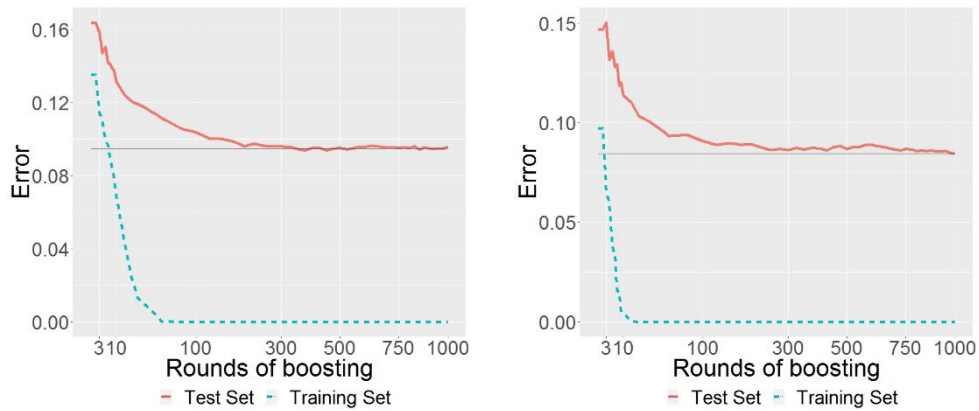
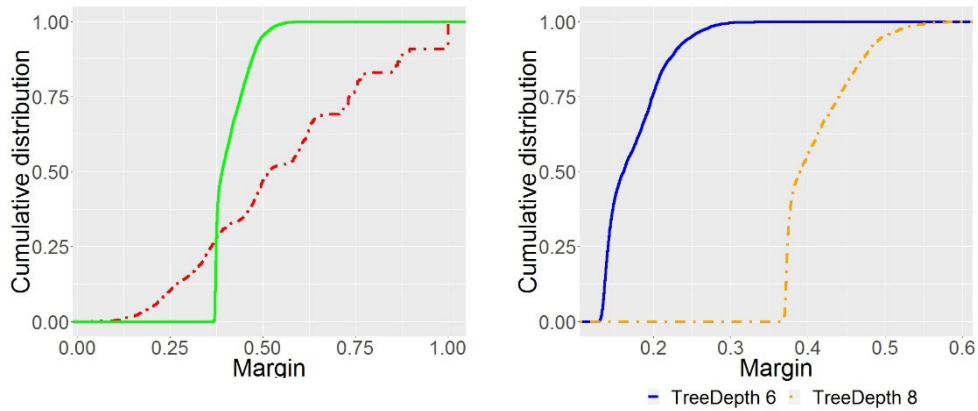


Figure 3 Left-hand side: margin distribution of the training cases after 15 and 1000 rounds. Right-hand side: comparison of margin distribution using different tree depth and 1000 rounds.



4. Noise robustness

In this section we discuss AdaBoost as a "spiked-smooth" classifier with a noise robustness property. Wyner et al. (2017) define a spiked-smooth algorithm as one that fits the noise points only in an extremely local neighborhood, hence shrinking down their negative influence on the accuracy of the classifier.

We will use the phoneme dataset, which was described in Section 2. Noise points were generated by changing the labels of 100 points in the training set. To obtain more general results, the procedure was repeated five times and the results were averaged.

The performance of AdaBoost with maximum trees depth of eight was compared with random forest and 1-nearest neighbor algorithm. For the phoneme dataset, all three algorithms considered are interpolating classifiers (Wyner et al., 2017) – they assign a correct class label to every training set observation.

Table 1 Model accuracy error rate on test set.

Dataset	AdaBoost 300 Rounds	AdaBoost 1000 Rounds	Random Forest	1-NN
phoneme	0.0986	0.0984	0.0980	0.1011
phoneme with noise	0.1036	0.1012	0.1020	0.1248
difference	0.0050	0.0028	0.0040	0.0237

Even after a significant number of points' signs in the training set were flipped, both AdaBoost and random forest results did not decrease significantly, especially compared to results of 1-NN algorithm (Table 1). Note that the similar behavior of AdaBoost and random forest could be attributed to the similar structure of these algorithms, as they both in some way average results of other classifiers.

The results for AdaBoost with 300 rounds and 1000 rounds suggest that with an increasing number of iterations the noise points influence a smaller number of correct points. In other words, the fit is more local.

5. AdaBoost variants and other boosting approaches

5.1 Modifications for multiclass and soft classification

So far only the hard classification case was taken into consideration, that is, output values were only considered as either -1 or +1. There exist some extensions to the presented methodology that provide probability estimate of class like LogitBoost (Friedman et al., 2000) which is based on minimizing loss function:

$$E = \sum_{i=1}^{N} \ln(1 + e^{-y_i H_m(x_i)})$$

In this case, the only demanded change to regular AdaBoost is redefining $w_m(i)$ to be proportional to $\frac{1}{1+e^{y_i H_m(x_i)}}$.

As for the multiclass classification case, usually the "one vs. all" technique is used. It consists of taking a single class as a first label and all the other possible classes as the second label.

5.2 AdaBoost for regression

Just as decision trees can be used for regression (targets in splitting nodes are not 0–1 but rather continuous values), AdaBoost can be used for regression applications. This setup is very much like classification in terms of weighting samples that have higher error on previous classifiers. What changes is the loss function that determines this and the likelihood of continuous rather than binary values in y variable (Ridgeway, 1999).

5.3 Gradient Boosting – modern boosting approaches

After first successful implementations of boosting, other techniques based on its assumptions emerged. A technique called gradient boosting shows unsurpassed results on a variety of datasets. The prominence of these methods is evidenced in the popularity of algorithms like XGBoost (Chen, 20016), Catboost (Prokhorenkova, 2018) and LightGBM (Ke, 2017) in Kaggle competitions. These algorithms use different tricks and optimization techniques, but their idea is the same. Instead of optimizing the choice of a next classifier by weighting samples, gradient boosting takes advantage of computing gradient of error function with respect to the strong classifier at each step. The new classifier is then fitted to this so-called pseudo-residual which permits it to achieve even more bias reduction compared to Adaptive Boosting. This family of algorithms is much harder to fine-tune, though, meaning that AdaBoost often remains the default solution for many classification problems.

References

[1] Freund Y. and Schapire R. E. *"A decision-theoretic generalization of on-line learning and an application to boosting"*. 1997. Journal of Computer and System Sciences. 55: 119.

[2] Friedman J., Hastie T. and Tibshirani R. "Additive logistic regression: a statistical view of boosting". *Annals of Statistics*. 2000. 28 (2): 337–407.

[3] Geurts P., Ernst D. and Wehenkel, L. „Extremely randomized trees". *Mach Learn* (2006): 63.

[4] *Heart-disease*. https://archive.ics.uci.edu/ml/datasets/heart+Disease.

[5] Ke G. et al. "LightGBM: A highly efficient gradient boosting decision tree". *Advances in Neural Information Processing Systems* (2017): 3149–3157.

[6] Leshem G. "Improvement of AdaBoost algorithm by using random forests as weak learner". Ph.D. Thesis. 2005. Hebrew University of Jerusalem.

[7] Lienhart R, Maydt J. "An extended set of Haar-like features for rapid object detection". *Proceedings. International Conference on Image Processing*. Rochester, 2002.

[8] Maclin R. and Opitz D. "An empirical evaluation of bagging and boosting". *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*. Rhode Island, 1997. 546–551.

[9] Mahmood Z., Ali T. and Khattak S. "Automatic player detection and recognition in images using AdaBoost". *Proceedings of 2012 9th International Bhurban Conference on Applied Sciences & Technology (IBCAST)*. Islamabad, 2012. 64–69.

[10] Papageorgiou C. P., Oren M. and Poggio T. "A general framework for object detection". Bombay: Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271), 1998. 555–562.

[11] *Phoneme*.https://www.elen.ucl.ac.be/neural-nets/Research/Projects/ELENA/databases/REAL/phoneme/phoneme.txt.

[12] Prokhorenkova L. et al. "Catboost: unbiased boosting with categorical features". *https://arxiv.org/pdf/1706.09516.pdf.* (2018).

[13] Ridgeway G., Madigan D. and Richardson T. "Boosting methodology for regression problems." *AISTATS* (1999).

[14] Schapire R. and Singer Y. "BoosTexter: A boosting-based system for text categorization". *Machine Learning* (2000): 135–168.

[15] Schapire R. E. "Explaining AdaBoost". *Empirical Inference*. Springer, 2013.

[16] Schwenk H. and Bengio Y. "AdaBoosting neural networks: Application to on-line character recognition". *International Conference on Artificial Neural Networks*. Lausanne, 1997. 967–972.

[17] Shapire R. E. et al. "Boosting the margin: A new explanation for the effectiveness of voting methods". *Ann. Stats.* 26.5 (1998): 1651–1686.

[18] Chen T. and Guestrin C. "XGBoost: A scalable tree boosting system". *arXiv* (2016).

[19] Thongkam J. et al. "Breast cancer survivability via AdaBoost algorithms". *HDKM '08 Proceedings of the second Australasian workshop on Health data and knowledge management*. Wollongong, 2008. 55–64.

[20] Tieu K. and Viola P. "Boosting image retrieval". *Proceedings IEEE Conference on Computer Vision and Pattern Recognition.* Hilton Head Island, 2000. 228–235.

[21] Viola P. and Jones M. "Rapid object detection using a boosted cascade of simple features". *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Kauai, 2001.

[22] Wyner A. J. et al. "Explaining the success of AdaBoost and random forests as interpolating classifiers". *Journal of Machine Learning Research* 18 (2017): 1–33.

[23] Xuchun L., Lei W. and Sung E. "A study of AdaBoost with SVM based weak learners". *IEEE International Joint Conference on Neural Networks*. Montreal, 2005. pp. 196–201 vol. 1.

[24] Zhang C. "Boosting-based multimodal speaker detection for distributed meeting videos." *IEEE Transactions on Multimedia* (2008): 1541–1552.

About the authors

I hold a MSc degree in Computer Science as well as BSc in Control Engineering and Robotics which I obtained at Wroclaw University of Science and Technology and AGH University of Science and Technology, respectively. I started my adventure with Nokia in 2017 by becoming a Working Student Data Analyst. Since 2018 I have been employed as Data Scientist in CMBD AI Lab Team. I try to combine the technical experience in modern AI and ML methods with business skills, both at work and as my hobby.

Maciej Kondrat
Data Scientist
NOPS CMBD CSBD AI Lab

I received my MSc degree in mathematics at Wroclaw University of Science and Technology in 2017, with specialization in Statistical Mathematics. I started my adventure with Nokia in 2018 by becoming a CMBD AI Lab Team member. I make sure to keep up with ML and AI news and innovations.

Paweł Skoliński
Data Scientist
NOPS CMBD CSBD AI Lab

I received the B.Sc. degree from the AGH University of Science and Technology in Kraków, Poland, and the M.Sc. and Ph.D. degrees from the University of Agder in Grimstad, Norway, in 2011, 2013 and 2016, respectively, all in mechatronics. In 2015, I was a visiting Ph.D. student in the Automatic Control Laboratory, ETH Zürich, Switzerland.

Since 2018 I have been with Nokia, Wrocław, Poland, where I am currently leading the Artificial Intelligence Laboratory by CMBD. My research interests include data science, optimization techniques, robotics and automation, AC drives, control systems, and modeling and simulation.

Witold Pawlus
AI Lab Lead
NOPS CMBD CSBD AI Lab

Feature Selection techniques

Marcin Koralewski
Data Analyst
GS CS Network Engineering

Adam Jankowiak
Specialist, Software Development
GS CS Network Engineering

NOKIA

Machine Learning (ML) is an enormous set of algorithms and concepts, which helps in giving insights about company business. But all ML models are based on input data, therefore their quality depends on quality of data. Each data (e.g. information about single BTS) is described by many features – part of them can be not important in specific case. We want to filter them, so we will get only significant variables. Without help of technical specialist, it is a task that requires support of machine learning methods.

1. Introduction

Let us start this overview of techniques by simple example:

Let model of data be:

$$y = X\beta + \epsilon,$$

where β is a sparse vector. That means that we are trying to estimate vector $\hat{\beta}$ so later we can estimate $\hat{y}$ as follows:

$$\widehat{y_i} = \widehat{\beta_0} + \widehat{\beta_1} x_{i1} + \cdots + \widehat{\beta_n} x_{i1},$$

where some of $\widehat{\beta_i}$ are equal to 0. When $\widehat{\beta_k}$ is equal to 0, that means k-th feature is insignificant and may be skipped in creating model – this is what we call Feature Selection.

One of the most popular indicators of model quality, which allows to compare models based on different number of variables is AIC [*Akaike Information Criterion]:*

$$AIC = -2 \sum_j \log(\hat{\pi}_j) + 2q$$

If we have had all possible models, then we could (using AIC) choose the one that works best (taking into account number of features in it). But, checking all 2^N models is often not available due to computation time. Therefore, there is a strong need of a method that will provide as good results as possible.

2. Advantages of FS

Sometimes Feature Selection is the first and only goal in determining what features are important – one may want to know what features affect modelled variable. For example, one may want to know which KPI's history affects operator's will to buy new software the most.

But sometimes we use FS because we want to:

- make model easier to interpret, simplify it,
- reduce *overfitting*,
- shorten computation time.

Nowadays, when data is easily accessible, two major problems in creating ML models that concern model quality are overfitting and curse of dimensionality. By overfitting we mean situation when there are too many dimensions when compared to amount of data. To get intuition why "overfitting is very dangerous", an active reader may now decide how much would he trust a model if it was created basing on:

- 700 data and 10 dimensions,
- 10 data and 10 dimensions,
- 5 data and 10 dimensions.

Rule of thumb says that we need at least 5–10 sets of data for each dimension.

Expression *dimensionality curse* is very broad and hard to define, but it is mostly understood as situation when there are too many dimensions in model. Let us assume that our data is uniformly distributed in a d-dimensional 0–1 cubic square. Let us assume that we are creating some local estimator, which takes information from 10% of data. When d is equal to 1, we just take 10%. But when d is equal to 2, we have to take 31.6% of each dimension. What happens when the number of dimensions becomes really high?

Figure 1

1-D: 10% of dimension taken to get 10% of all data

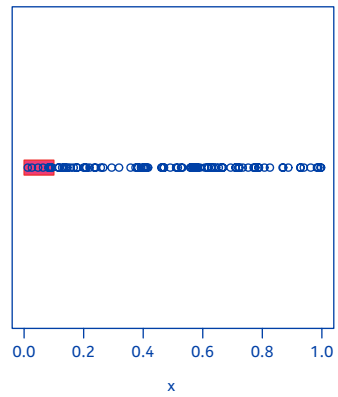
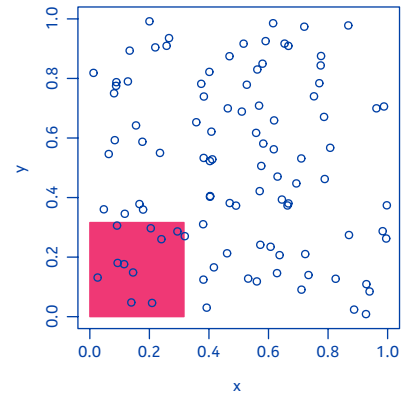


Figure 2

2-D: 31.6% of each dimension taken to get 10% of all data



When there are 100 dimensions, to take 10% of data, one may have to take 97.7% of each of dimensions! That does not come with idea of a local estimator anymore.

When there are too many dimensions in data, also computation time is becoming larger.

3. Randomization

Let us come back to problem defined in the introduction: there exist data created by some model (linear regression), when we know that some of data points' features are irrelevant and should not be taken into account by model. Unfortunately, we cannot check all possible combinations of features to decide which ones are proper. Therefore, we will introduce some algorithms that may help in such situations.

Before we start, it may be useful to get familiar with some vocabulary. By Monte Carlo algorithm we define an algorithm which returns result in a short time, but the result may be not optimal. By Las Vegas algorithm we define an algorithm that returns optimal result, but computation time may be really long.

3.1 Stepwise regression

Stepwise regression, despite its weaknesses, is one of the most popular ways to determine which features should be taken into linear regression model. Although it is not a randomized method, but a greedy algorithm, one may not say that he is familiar with feature selection techniques without knowledge about this one.

Listing 1 Forward stepwise regression

```
Data:  X = <x_1,y_1>, …, <x_n,y_n>
crit ← Criterion of adding variable
Result: Set of chosen features: X_best
Begin
feat_best ← empty set of features
model ← linear regression model based on feat_best
              features
for i=1 to N {
  best_model ← model
  for feat in (features – feat_best){
    model_pretend ← model based on feat_best
                           features + feat
    if crit(model_pretend) > crit(best_model){
      best_model ← model_pretend
    }
  }
  if model = best_model{
    break
  }
  model ← best_model
  feat_best ← features of model
}
End
```

As it may be easily noticed, stepwise regression is a basic greedy algorithm, which may not find optimum set of features. Big advantage of this method is short computation time and its simplicity.

In this method we use some criterion to decide which feature is to be added to the model. There are many criterions that may be used in this situation:

- AIC
- R^2, adjusted R^2
- F-test, t-test

Complex comparison of these methods is out of scope of this article, but on this point adjusted R^2 is one of the most popular methods to evaluate quality of a linear regression model.

3.2 MC1 algorithm

Not every randomized algorithm has to be based on a model of data. There is also an important group of *filter methods*, which try to drop a number of features so only the most important ones are kept.

Decision which features are to stay is always non-trivial. There are a few concepts that are popular when we are using filter methods. Our decision may be based on:

- Variance within dimensions (with assumption that variance is bigger when feature gives more information/is more important),
- Ability to separate classes,
- Ability to keep the structure of data.

The concept of MC1 algorithm is based on keeping the local structure of data. For a subset of features there are created a training data set and a test data set. On the test data set we are checking (using k-Nearest Neighbor algorithm) what the accuracy of estimation is. If it is high, that means that the local structure of data set is kept.

Listing 2 MC1 algorithm pseudo code

```
Data:  X = <x_1,y_1>, …, <x_n,y_n>
t_max = number of iterations
p – number of features
Result: Set of chosen features: X_best
Begin
X_best ← set of randomly chosen p features
for i=1 to t_max {
  X_prim ← random p features
  if kNN(X_prim, X) > kNN(X_best, X){
    X_best ← X_prim
  }
}
End
```

Before we proceed: whether MC1 algorithm is Monte Carlo or Las Vegas algorithm depends only on t_{max} parameter. Of course, this algorithm is not efficient when treated as Las Vegas algorithm due to computation time. As a result of this filter method there a subset of p features is returned, where p is determined as an input.

3.3 Relief algorithm

Another interesting example of filter method is relief algorithm, whose concept is based on ability to separate classes. This is a simple algorithm, used when we have only two classes of data.

Main idea of the algorithm is to count distance from each of points to the nearest point in both classes in each of dimensions. When classes in one of dimensions are well separated, then distance in this dimension is high and the feature described by this dimension is treated as important.

First, we have to define a procedure that will be later used to count distance from a point to the nearest points in both classes:

Listing 3 Procedure update (w, x_i, x_j^+, x_j^-)

```
Begin
for i=1 to N {
  w_i ← w_i – diff(x_i,x_j^+)² + diff(x_i,x_j^-)²
}
return w
End
```

When we already have procedure of updating w, which is a vector of differences of distances between classes within dimensions, we can proceed to the main part of the algorithm:

Listing 4 Relief algorithm

```
Data:  X = <x_1,y_1>, …, <x_n,y_n>
t_max = number of iterations
τ = significance level
Result: Set of chosen features
Begin
Split X into X^+ and X^–
w ← (0,0, …, 0)
for i=1 to t_max {
  x_i ← random example
  x_j^+ ← closest to x_i example ∈ X^+
  x_j^- ← closest to x_i example ∈ X^-
  if x_i ∈ X^- {
    w ← update(w, x_i, x_j^+, x_j^-)
  }
  else {
    w ← update(w, x_i, x_j^-, x_j^+)
  }
for i=1 to N{
  if w_i / t_max ≥ τ {
    Feature i-th is important
  }
}
End
```

As a careful reader may notice, in this algorithm we do not count distance to both classes for each of points, but only for random subset. That means this is a Monte Carlo algorithm.

If we want to compare two introduced filter algorithms (MC1 and Relief algorithm) we may notice that Relief algorithm is much faster due to number of operations that have to be executed (mostly distance counting). However, it is more vulnerable to random choice of data to learn and as an algorithm it is interesting mostly for its concept of dimensions' ability in class separating.

3.4 Simulated annealing

Simulated annealing is an interesting and powerful randomization algorithm. Its name came from annealing in metallurgy and knowledge about this process gives good intuition how the algorithm works. When we anneal metal, we want atom structure to become organized, even though on start it was highly unorganized. Metal is being heated to high temperature, and atoms start to migrate afar in search of a strong, stable configuration. As the time passes, metal is cooling and atoms do not get enough energy to continue such greedy search – they start searching locally. As metal is becoming cooler and cooler, local search for structure takes place on a smaller distance. After the process ends, metal is less hard and more workable – that is because some optimum of metal structure was found by its atoms.

We can easily use logic of local search of optimum for feature selection case. However, first we have to define what is an analogy of atom and distance.

Of course, as an atom we treat our solution – series containing zeros and ones which stands whether $feature_n$ is treated as an important feature or not. This may be visualized as:

Table 1 Sample series of feature importance

	Feat 1	Feat 2	Feat 3	Feat 4	Feat 5
Scenario 1	0	1	1	0	1
Scenario 2	1	1	1	0	1
Scenario 3	1	0	1	0	0

So, when we know how our *states*, between which our algorithm will search in hope of finding the global optimum, we have to define what *distance* is. This term is important because the range of search will become smaller and smaller as the temperature becomes lower.

Fortunately, in mathematics counting distance between two series is a well-known problem. As the series contains 0–1 variables, the most natural measure of distance will be the l_0 norm.

This norm is counted as number of features on which series make different decisions whether to treat features as important or not.

Listing 5 Simulated Annealing in feature selection

```
Data:  X = <x_1,y_1>, …, <x_n,y_n>
Annealing schedule, T_0, T_final and ΔT
eval ← function to evaluate quality of proposed solution
neighbor ← function to search for neighbor of solution
           provided, depending of temperature of system
Result: Set of chosen features: X_best
Begin
S_best ← random subset of features
While T_i > T_final {
  S_i ← neighbor(S_best, T_i)
  ΔE ← eval(S_best, X) – eval(S_i,X)
  if ΔE < 0{
    S_best ← S_i
  }
  else{
    S_best ← S_i with probablity exp(-ΔE/T_i)
  }
  T_i ← ΔT × T_i
}
End
```

As we can see in pseudocode, the idea of simulated annealing algorithm is really close to annealing used in metallurgy. We start from a random solution (decision what features are important), evaluate how well this solution works, and then start searching for a better one. When the system has high *temperature*, we are looking for solution completely different from the one we have and hoping that it may be better. As the system is cooling, we look for a solution which is more familiar – as our solution is good, we hope that change of one feature may make it better, but we do not want to take risk and change everything.

4. LASSO & Dantzig Selector

Algorithms we were talking before mostly focus on idea of randomization (except stepwise regression, which was a greedy algorithm) and evaluation of solutions provided. Another idea of constructing a feature selection technique is to create a model and combine profit from well fitted model with penalty for model complexity. Such concepts in mathematics are also called *regularization*.

For linear regression model, when the y is defined as

$$y = X\beta + \epsilon,$$

we define LASSO (least absolute shrinkage and selection operator) as:

$$\min_{\beta} \|y - X\beta\|_2 + \alpha \cdot \|\beta\|_1$$

As we can see, in LASSO we try to choose β so at the same model we get small residuals and sparse estimator of β – sparsity of estimator is caused by penalty for size of β in l_1 norm. LASSO is easily generalized – one of the most popular usages is LASSO for logistic regression.

One of the greatest opponents of LASSO is Dantzig Selector – a regularization technique, often used in compressed sensing to restore sparse signals which were highly compressed.

In feature selection techniques Dantzig Selector almost always gets better results than LASSO. Its idea is slightly different. In Dantzig Selector, we require residuals to be non-correlated and small and then we try to find β, which fulfils these requirements. Formally, it is denoted as:

$$argmin_{\beta} \|\beta\|_1, \text{ when } \|X^T (y - X\beta)\|_{\infty} \leq \delta$$

These two regularization techniques are very popular in feature selection problems, especially in linear regression model. We cannot forget that these are just definitions of regularization problems, not valid algorithms which will give us solution as a β vector. Despite the fact that both these methods are implemented in statistical software, solving these problems is numerically hard and sometimes requires the same meta-heuristics we were talking before, like simulated annealing.

References

[1] L, Huan, Computational Methods of Feature Selection, 1958.

About the authors

I have graduated my master studies in Mathematics in 2018, with specialization in Statistical Mathematics. My passion is discovering new Machine Learning algorithms with their pros and cons. I work for 2+ years in Network Engineering, where I help in providing solutions that require algorithm design and implementation of ML techniques.

Marcin Koralewski
Data Analyst
GS CS Network Engineering

I have graduated my bachelor studies in Mathematics in 2016 and currently I am pursuing my master's degree in it with specialization in Statistical Mathematics. I explore subjects on survival analysis. I have been working for a year in Network engineering, where I contribute to data analysis and algorithms design.

Adam Jankowiak
Specialist, Software Development
GS CS Network Engineering

Extreme Learning Machine: The Extremely Fast Neural Networks

Tomasz Szandała
Software Configuration Engineer
MN CDS Software Configuration Management

NOKIA

Has any of you ever tried to train a really deep neural network using only CPU? Yes, it might take ages, mostly if you lack a current generation GPU. Luckily there is an alternative option: a neural network that learns even 70 times faster than a standard deep neural network.

1. Extreme learning machine

It is clear that the learning speed of feedforward neural networks is in general far slower than the satisfying speed, which is considered as a major bottleneck in its applications. The key reason: the slowness of backpropagation learning algorithms which are extensively used to train neural networks. Backpropagation is a process in which all parameters of the networks are modified iteratively by using backpropagation learning algorithms. Unlike these traditional solutions, a new learning algorithm called extreme learning machine (ELM) for single- and multi-layer feedforward neural networks was proposed by a Chinese professor Guang-Bin Huang. Those networks randomly choose the input weights and internal weights (between hidden layers), and finally analytically determine the network's output weights. In theory, this algorithm tends to provide very extensive generalization performance along with an extremely fast learning speed. The empirical researches based on real-world problems solving show that the described algorithm can compete with traditional, popular learning algorithms for feedforward neural networks.

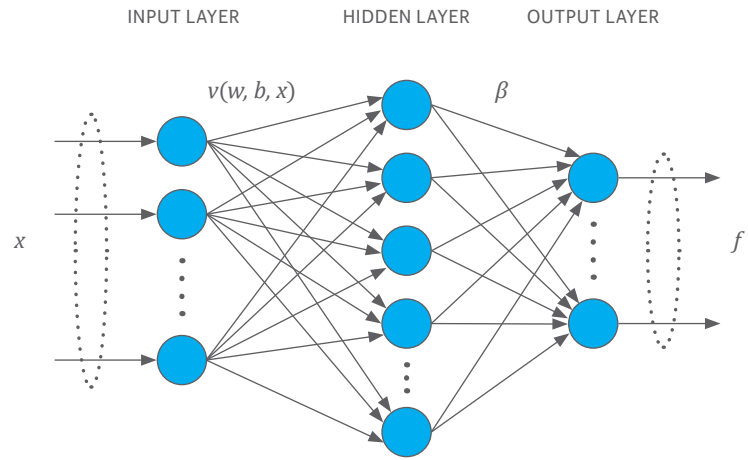
In the 1940s, mathematician Walter Pitts and psychologist Warren McCulloch described a first artificial neural neuron which preluded the entire artificial neural network research. Neural networks turned out to have a strong non-linear mapping ability and adaptive self-learning, robustness, and fault tolerance characteristics. Perfect solution for the age of the information amount of data.

It is said that ELMs fill the gap between Frank Rosenblatt's Dream and John von Neumann's Puzzle [2]. It is a simple architecture of an artificial neural network, typically consisting of three layers:

- Input layer that collects input data
- Hidden layer (usually one, but there are no real obstacles to use more)
- Output layer that collects signals from a hidden (internal) layer and gives network's a final verdict

The only difference between ELM and a classic network is one-time assigning weights to input connections in a hidden layer. All weights between the input and the hidden layer, and all weights between the layers in hidden neurons are randomly assigned when network is created, and afterwards those connections become frozen. Only weights in connections between the hidden and the output layer are influenced by the training process.

Figure 1 Conceptual diagram of ELM architecture with one hidden neural layer



2. ELM training algorithm

The simplest extreme learning machine (ELM) training algorithm, chosen for this paper, works in a single hidden layer network, with the sigmoid activation functions:

$$Y = W_2 F_h(W_1 X) \quad (1)$$

Where W_1 (eq. 1.) is the matrix of weights between the input and the hidden layer, fh is an activation function of the hidden layer, and W_2 is the matrix of the hidden-to-output-layer weights.

The training algorithm proceeds as follows:

1. Fill W_1 with random values, preferably from the range of [-1,1].

2. Estimate W_2 by least-squares fit to a matrix of response variables Y that are computed using the pseudoinverse $\cdot+$, given by a design matrix X (eq. 2.), while (eq. 3) shows how to compute the pseudoinversed matrix M^+:

$$W_2 = \mathrm{f}(W_1 * X)^+ * Y \quad (2)$$

$$M^+ = (M^T * M)^{-1} * M^T \quad (3)$$

Pseudoinverse has already been proven superior for some artificial neural networks training, for example Hopfield neural network for pattern recognition [4]. Computation of the pseudoinverse is much simpler, therefore uses less computing power than a whole backpropagation algorithm. Sample benchmark on MNIST 10 dataset shows over 70 times shorter time of training between deep belief

neural network (taught by backpropagation) and ELM network. While computing pseudoinverse matrix for the computer is very easy, be cautious since it might be difficult when we take into consideration high-dimensional images.

Unfortunately, one of the problems of ELM is that it does not mention anything about the data's dimension. And in most recent cases, it is almost impossible to compute with the entire data at once (because we might not have it all), therefore we must find a way of learning the output weights sequentially. While common networks learn each example one by one and can tune the network to new instances, it is not so simple in ELM. Of course, we can recompute the entire pseudoinverse enriched with new training examples, but this is not useful in the production environment where we would have to keep the training data with the network. There are solutions, like adding additional parameters to output the activation functions, but they introduce unnecessary complexity. Lack of expansion network with new possibility examples is one of the open problems for ELM.

3. Accuracy is still being worked on

While the speed of learning is unquestionable, there is still a factor of accuracy. Simple transition from common network to extreme learning machine (ELM) gives up to around 10% worse results for simple classification [3]. Recent papers show that it can be changed. For example sigmoid activation function into rectified linear unit (ReLU) function can diminish the difference to a single percent, and there is even a possibility that it could be eliminated. But science cannot rely on luck, we can make use of the genetic algorithms in order to enhance our network. Evolving of neural network is considered as extremely inefficient in case of computing. But does it still apply to ELMs?

My recent research origins from this question. Since in ELM we cannot change internal weights, I decided to evolve them in each generation and then use pseudoinverse, also in each generation.

I have used ELM to classify handwritten letters described by 16 attributes. The parameters for my algorithm were:

- Network: 16/80/26 nodes per layer
- Hidden neurons activation function: sigmoid
- Initial population: 40
- Mutation ratio: 2% mutation is a random reassignment of weight for a given connection
- Epochs: 30

Each network was a one specimen that was first evolved (by crossovering and mutation) and then, prior evaluation, trained using pseudo-inverse.

Figure 2 Genetic algorithm schema

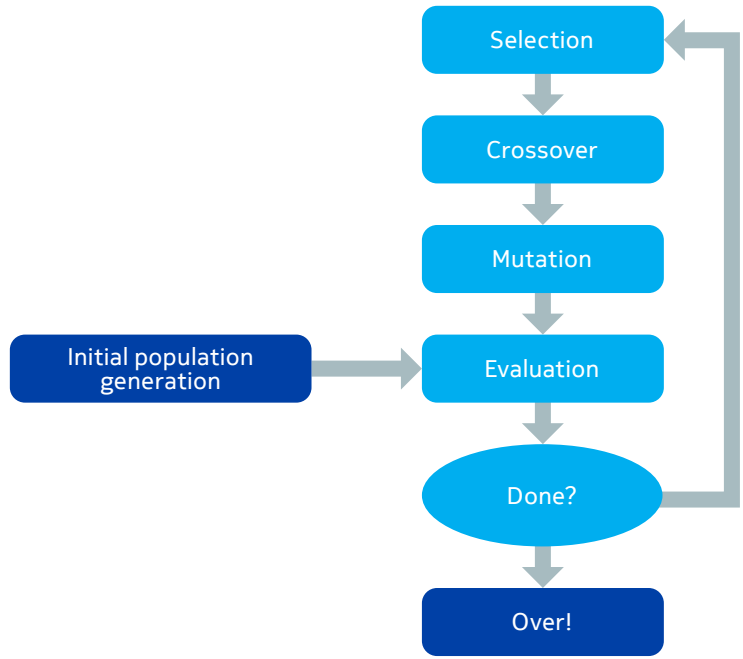


Table 1 Experiment results for three sessions and their average values

	Attempt I	Attempt II	Attempt III	Average
Time of the entire process [mins]	12:36	11:58	11:54	12:09
Accuracy of the best specimen from the initial population	0,69	0.65	0,70	0,68
Accuracy of the best specimen post-GA	0,73	0,70	0,72	0,72

As we may see, the time of 30 evolutions for 40 specimens (so we trained 1200 networks) was low despite being executed on CPU. Furthermore, in each attempt the network after evolution has been more accurate in its classification task. What we can also conclude is that ELM itself is quite a useful tool. The accuracy was rather stable even during 30 generations. There is no big gap in performance between the first and the last epoch. This means that ELM does not rely so much on inner weights but mainly on the pseudoinverse ones.

4. Summary

To sum up, extreme learning machine (ELM) appears to be a noteworthy competitor to common, well-known deep learning. While it is fast, it also keeps high correctness in results. The utilization of genetic algorithm (GA) improves its efficiency for, at least for now, a small degree and it may be improved even further. While I focused on simplicity of the example, it is worth to for example extend the amount of training epochs or increasing the number of neurons in the network in order to achieve better results.

Regarding drawback, it is very difficult to scale up ELM network, especially for data like high-resolution images. For millions of images, the system is going to compute between very large matrices, so the training time can grow rapidly. The second disadvantage is that the teaching the network new data is virtually impossible for ELM. Unlike common networks, we would have to fully retrain our model in ELM instead of only teaching new examples.

Taking into account the pros and cons, we can consider ELM as an interesting alternative to commonly used deep neural networks, that has a potential to fill some unique gaps in the discipline.

References

[1] Guang-Bin Huang, Qin-Yu Zhu, Chee-Kheong Siew, Extreme learning machine: Theory and applications, Neurocomputing 70, 2006.

[2] Guang-Bin Huang, What are Extreme Learning Machines? Filling the Gap BetweenFrank Rosenblatt's Dream and John von Neumann's Puzzle, Cognitive Computing, 2015.

[3] Guang-Bin Huang, Hongming Zhou, Xiaojian Ding, Rui Zhang, Extreme Learning Machine for Regression and Multiclass Classification, IEEE Transactions on Systems, Man and Cybernetic, 2011.

[4] Tomasz Szandała, Comparison of Different Learning Algorithms for Pattern Recognition with Hopfield's Neural Network, Procedia Computer Science, Volume 71, 2015.

[5] Back Thomas, Evolutionary Algorithms in Theory and Practice, 1996.

About the author

I am a PhD student at Wroclaw University of Science and Technology in the field of Artificial Intelligence. I am also a member of the Creation & Development Group KREDEK, which is a university science club. In NOKIA I work as a Software Configuration Engineer in Wroclaw's DevOps Team. My daily work consists of deploying and maintaining hundreds of hosts for multiple purposes: Jenkins, web applications, compilation servers, and so on. My favorite task in life is solving problems since with computers impossible is nothing, it just takes a bit more time.

Tomasz Szandała
Software Configuration Engineer
MN CDS Software Configuration Management

Handling imbalanced classification problems

Karolina Herlender
Software Engineer
MN CDS Software Configuration Management

NOKIA

Machine learning scenarios with inequal classes representation often result with misleading classification accuracy. Many real-life cases can be considered imbalanced, if there is one or few dominating classes it may cause other to be ignored by learning algorithm. This article will guide with various approaches that allow to handle such learning scenarios. Techniques are compared on exemplary data with different classes disproportion to find performance metrics with most reliable results assessment.

1. Imbalanced data

Imbalanced datasets in machine learning classification problems are very common. Most of time systems are working properly while breakdowns happen only accidentally. Usually most of population are not suffering for some diseases. All routine situations with exceptions are perfect examples of what imbalanced data is.

Imbalanced data are exactly such cases when the classes are not represented equally and have a meaningful difference in classes ratio. Class is a group of instances labeled to be recognized after model processing. Inequality ratio is simply the proportion of instances of each class. With 100 samples that represent class A and 10 samples that represent class B proportion is 100:10, so 10:1. In this example class A is called majority class and class B minority class.

Much focus in machine learning area is on building classification models. With different algorithms data is processed, analyzed and each row of data is labelled by class that it belongs to. There are many techniques to build for example predictive model or system that helps with expertise. To teach the machines, datasets must be provided. Based on training data in the future model will be able to classify data that seems like those in training examples to the most similar class.

Algorithms used in machine learning can face the data sets with not equal number of samples in class but with those exceptional situations may not work well. As a result, with imbalanced data, prediction accuracy may be very high, but without any special techniques to deal with imbalanced data it may happen that all data belongs only to majority class.

Gathering more data improve predictive models but it is often not possible, because the situations like fraud transaction in bank, hard disk failure or server breakdown are not often and there is a need to wait very long time to register enough cases and to collect new samples. Even with larger datasets inequality ratio can be still the same or be growing too.

The purpose of this paper is to show possibilities to handle imbalanced data. To understand the examination below are described method that helped to improve model accuracy.

2. Cure for imbalanced data

2.1 Dataset proportions

The most common ways to process with imbalanced data are involved with artificial data amount manipulation. Such methods are undersampling (deleting rows from majority class) and oversampling (generating rows from minority class).In this paper there will be comparison for different datasets to answer which method can be better for given classification problems. In general the more balanced dataset is the better results for the classification model. The final evaluation will be proceeded by measure AUC (Area Under the Curve).

2.1.1 Undersampling majority

Undersampling is the method where the observations are removed. The idea to help with imbalanced data is to reduce samples from majority class drifting to equality between classes. In implementation it is a filter that passes minority samples and randomly filters rows from majority class. After applying the filter datasets are balanced, but how about results? Many machine learning models require large datasets, the error rate of classification is minimizing by evaluation with existing label. For example if it is very little amount of data from minority class, just a few rows, the whole dataset may be not enough to teach the model.

2.1.2 Oversampling minority

Opposite to undersampling are methods involved with oversampling. Balance in classes can be achieved when dataset with minority class is growing. Most common algorithms for oversampling are Bootstrapping and SMOTE.

Reduction of data may significantly harm the classification results if the number of rows is insufficient to properly train the model after reduction. If reduction is insufficient to face imbalanced data then data addition could be considered. Instead of removing samples from majority dataset, minority dataset can be enlarged. New rows are created based on algorithms that rely on statistics. The approach is called oversampling: creating new samples increasing primary dataset. Below will be presented two most popular oversampling techniques: bootstrapping and SMOTE.

In statistics bootstrapping relies on random sampling with replacement. Observations from the minority class are randomly sampled. Since replacement is allowed some rows can be duplicated many times and other ones omitted.

SMOTE is acronym from Synthetic Minority Oversampling Technique. This method was created when the authors realized positive effects in classification when they took into consideration area in decision space. In general algorithm oversamples minority datasets by forming new synthetic samples based on the feature space generated by samples in minority class. SMOTE uses algorithm of k-nearest neighbors to consider the features to the new samples from neighbors. This oversampling method tries to initiate new data similar to neighboring samples. Some features that are significant to class affiliation created the feature space. Observations from minority class are sampled and entered to synthetic example along the features combined k nearest neighbors. As an effect there are formed new artificial samples that allow for datasets growing.

SMOTE can be executed with different parameters for k-neighbors and with different oversampling quantity, in how high rate current dataset must grow. Depending on the number of samples needed to reproduce another number k for k-nearest neighbors gives better classification effects. Synthetic samples are generated by consideration of the difference between vector of features and nearest neighbors. The difference is multiplied by a random number within a range from 0 to 1 and added to the vector of features. This allows to randomly decide for a date between the lines of segmentation between two different specific features

2.2 Variable selection

In machine learning data extraction, construction and selection are techniques that allow transformation and simplification of datasets to make classification more accurate. A correlation between features is examined and then various techniques can be applied to improve classification.

Selection can remove the features that are irrelevant in classification process. Construction is an opposite process. Feature extraction is finding the feature representation in feature space.

Rank methods are one of the easiest and the most common methods in feature selection. The rank is formed with the establishment that features are independent from each other.

2.3 Choosing the best algorithm in building prediction

Worth to consider is changing algorithms used to build data prediction model in machine learning. Some algorithms in machine learning would perform better on some datasets than another. Usually researches are not applying many models and leave the one with the best results of classification.

For example, in ML a very common method is Naive Bayes, which is used to build prediction models. and is used to build prediction models.

Naive Bayes is easy to apply and performs well for large datasets. Algorithm counts the frequency of features observations and creates likelihood by calculating the probabilities. Next step is calculating a posteriori probability for prediction.

For current experiment three algorithms were used: Naive Bayes, logistic regression, decision tree.

Logistic regression is an example of predictive analysis method. It is a method for binary values of variables. That means that variables inputs are combined linearly like in linear regression but with binary values. Regression is the method of the best matching of math formula to the given data to create the most accurate model. Regression model is formed mathematically based on probability of event belonging to vector of x variables. In the model exist two unknown parameters a and b that must be estimated from datasets. Model can be used to classify the observations for two classes at least. Probability of membership to one of the classes is always between 0 and1.

Decision tree is a hierarchical structure: it contains leaves, branches and nodes. It is a consistent acyclic directed graph. In decision tree branches are used to conduct decisions on features values and leaves store the results of the tests executions. Nodes are connected by branches and last nodes are leaves, where class labels are being held. In nodes are stored tests that are functions that transform observations into set of test results. Tests are executed on feature values of the observation.

3. Experiment execution

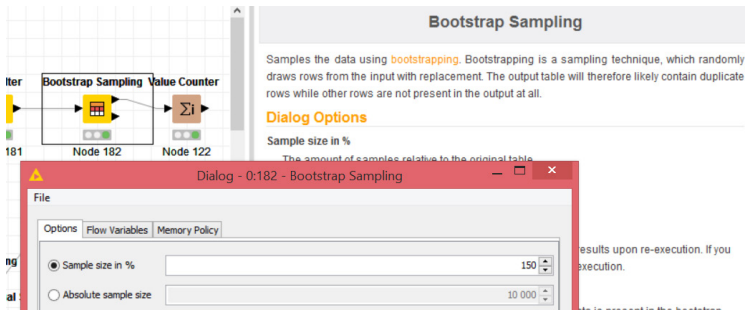
3.1 Environment

The experiment was executed in KNIME in version 3.2.1, software dedicated for analyzing and reporting data. KNIME allows to build graphical flow of data processing by using the canvas and icon to drag and drop and configure. Each “icon” is a node in the flow and executes the tasks in description with set configuration. Output preview is possible.

3.2 Datasets

The source of datasets was website kaggle.com. Kaggle is the community where people can share their interests for machine learning.

Figure 1 KNIME workspace with nodes, node description and configuration



Chosen sets contain the data with different imbalanced proportion. For each dataset preprocessing was executed were missing above 15% was removed, another missing value were filled by average values. Each time when model training was executed data was transformed to cross validation.

Below the datasets from Kaggle. Datasets were chosen to obtain diversity of datasets characteristics to examine and determine the best method for prediction depending on given set of features: number of features in dataset, number of observations and imbalanced rate.

Dataset name	Number of features	Number of observations	Rate of imbalance
bank_loan	20	5000	2,34
biddings	88	100000	523,11
candidate	17	1811	2,85
caravan	85	9822	15,76
credit card	23	30000	3,52
flights	19	100000	63,74
spaceX	15	41	4,13
speed_date	110	8378	5,07
spine	12	310	2,1

3.3 Results

In machine learning are many ways to build a classification model. To determine which model gives the best results there are methods for model assessment.

In this article the model performance was measured for each classifier methods in AUC. AUC abbreviation origins from area under curve in the graph presenting classification performance. Below you can find the table with outputs of each classifier in each dataset after described base preprocessing:

Dataset name	Naive Bayes	Decision tree	Regression logistic
bank_loan	0.803892	0.934813	0.821946
biddings	0.49401	0.478528	0.423526
candidate	0.931795	0.968543	0.961852
caravan	0.694451	0.660122	0.738064
credit card	0.734267	0.664818	0.722641
flights	0.999991	0.963023	0.768626
spaceX	0.670455	0.666667	0.814394
speed_date	0.901054	0.982718	0.949106
spine	0.873095	0.791786	0.925524

For some of datasets like “flights” AUC was very high, but for mostly there is a space to apply techniques that can increase this number.

3.4 Results after techniques for imbalanced data

In Knime we created 4 models to manage with imbalanced data. Each model was tested with 3 algorithms. In this configuration we obtain 12 results in AUC for each dataset.

Below is the table with results. Measure is AUC. Each row contains name of dataset and 12 results that are in this order: Naïve Bayes (NB), Decision Tree (DT), Logistic Regression (LR)

3.5 Analysis

Comparing algorithms used for prediction, the best efficiency achieved Naive Bayes.

So which method that helps with imbalanced data is the best? There is no method that would achieve the best results in all cases. The efficiency of method depends on the datasets that were applied and the classifier. The highest AUC for most datasets were for models that used SMOTE method to handle imbalanced data.

SMOTE is an improved method of oversampling. It enlarges the dataset, so it gives better effect than undersampling. Every classifier needs data to learn therefore majority class reduction approach leads to disability to learn by classification through data shortage.

In case of Bootstrapping method, the best results were obtained independently from imbalanced rate. That means that during the experiment examinations with increasing minority class were conducted. Repeating observations two or three times provided to the best results, independent from how big the rate of imbalance was. In given examples accuracy has been increased but it shouldn't be interpreted as unequivocal and undivided way to solve classification problems.

Experiments allowed us to see that there is no simple answer which method is the best to handle imbalanced data. Much depends on datasets, chosen classifier and number of features in the sets. Methods presented in this article may be treated as indication, but the best results should be chosen only by practice

References

[1] N. V. Chawla, N. Japkowicz, A. Kołcz, Editorial: Special Issue on Learning from Imbalanced Data Sets,
[2] I. Guyon, A. Elisseeff, An Introduction to Feature Extraction,
[3] H. Liu, H. Motoda, Feature Extraction, Construction and Selection: A Data Mining Perspective, Second Printing 2001, 4-8
[4] I. Guyon, A. Elisseeff, An Introduction to Variable and Feature Selection,
[5] W. Chmielnicki, Efektywne metody selekcji cech i rozwiązywania problemu wieloklasowego w nadzorowanej klasyfikacji danych, rozprawa doktorska, Kraków 2012
[6] X. Chen, J. Jeong, Enhanced Recursive Feature Elimination, Sixth International Conference on Machine Learning and Applications, 2007
[7] K. Z. Mao, Orthogonal Forward Selection and Backward Elimination Algorithms
[8] for Feature Subset Selection, IEE Transactions on Systems, Man and Cybernetics – part B: Cybernetics, vol. 34 no.1, February 2004

	Feature selection			Undersampling			Smote			Bootstrapping		
Dataset	**NB**	**DT**	**LR**	**NB**	**DT**	**LR**	**NB**	**DT**	**LR**	**NB**	**DT**	**LR**
bank_loan	0.805053	**0.934813**	0.822685414	0,799	0,877	0,819	0,804	**0,981**	0,825	0,815	**0,997**	0,841
biddings	0.69916	0.561084	**0.788778**	0,605	0,425	0,403	0,492	**0,565**	0,535	0,815	**0,996**	0,942
candidate	**0.982379**	0.958389	0.965181	0,935	0,948	0,985	0,888	**0,965**	0,958	**0,983**	**0,983**	0,943
caravan	0.719333	0.665107	**0.743982**	0,693	0,686	0,714	0,698	0,577	**0,736**	0,769	**0,949**	0,823
credit card	0.735494	**0.736445**	0.722984	0,605	0,425	0,403	**0,736**	0,71	0,723	0,74	**0,832**	0,773
flights	**0.999987**	0.967648	0.782159	**1**	**0,999**	0,74	**1**	0,997	0,811	**1**	**1**	0,807
spaceX	0.721591	0.6875	**0.768939**	0,7	0,5	0,84	0,708	0,718	**0,79**	**0,991**	0,809	**1**

About the author

My work is automating tasks that calculate the performance metrics for all Mobile Network. Every day I face a huge amount of data needed to be transformed, analyzed and rocessed to get the Key Performance Indicators.

Karolina Herlender
Software Engineer
MN CDS Software Configuration Management

Word Embeddings

Kamil Szatkowski
Technical Lead Engineer
MN BOAM TOOLS / Rain

NOKIA

1. Introduction

When one starts working in the machine learning (ML) area, one may encounter that not all problems rely on data that is densely packed within the given hyperspace. This means that some of the problems can be solved using a relatively small number of dimensions but relying on a continuous value within a single dimension. On the contrary, other problems require much more dimensions because the value is not continuous but rather discrete.

On one hand, there are problems like computer vision, which uses pixel data to learn high order features that can be used to recognize objects. Such pixel data exists in hyperspace where its basic property – color - is mapped, most of the times, into a continuous value range between 0...1 inclusive. A single pixel in the image is commonly represented with one (gray scale) or three (RGB) colors, which gives one or three dimensions for a single pixel. The image is a two-dimensional matrix of pixels (defined by width and height) which defines their arrangement and relations with each other.

On the other hand, natural language processing (NLP) is a different problem area. Base entity in (NLP) is a word (it might for example be a character, but in most cases, it is a word). In comparison to computer vision, a word is like a pixel but with a different type of value, that is a discrete value, hence it is a rather sparse space. Multiple pixels in the image represent a meaning. The same situation is with words which get a meaning when they are in presence with other words forming sentences, documents, and finally a corpus.

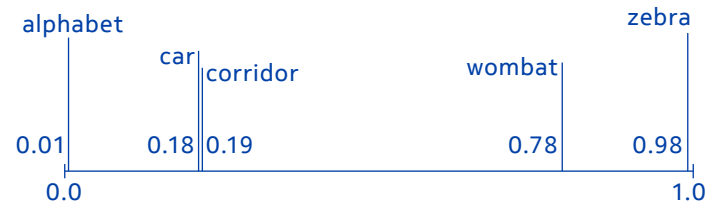
A corpus [1] is a set of documents that an ML algorithm is trained on. It should statistically represent relations between words that can be found in the language. There are two categories of corpuses – monolingual or multilingual. The first one is used to learn the relations within one language and the latter can be used to train translation algorithms. For this case such corpus is named a translation corpus and it contains the same text written in different languages. It enables the algorithm to figure out by itself which word in one language to replace with another word in the other language. Having that in mind, it is preferable to have a big and diverse corpus.

At a learning level, words are basic elements that algorithms are learned on – like pixels in computer vision, but there is a difference. Pixels can be described by a continuous range of values because each value represents the intensity of color for that pixel, for example 1.0 for a RED element means that a full power of RED is used and 0.95 means that it is still almost full RED (see **Figure 1**).

Figure 1 Red color intensity representation

However, the situation differs with words. Each word represents a different value. It is not useful to map 1000 words to the 0...1 space because a small error in value will convert to a totally different word with a completely different meaning. Even if the corpus is sorted, words that are next to each other may not have anything in common, for example CAR and CORRIDOR as in **Figure 2**. A different approach is taken with words – each word is treated as a different dimension. This means that if the corpus consists of 1000 unique words, each word has its own dimension. This changes the way the input to ML algorithms is supplied, and a one-hot representation is used for it to work.

Figure 2 Word corpus represented as 1D space



One-hot representation (or one-hot encoding) means that each word in an N-word dictionary is represented as an N-dimensional vector, so a Kth word has all elements that are set to 0 apart from the Kth that is set to 1 (**Figure 3**).

Figure 3 Example of one-hot encoding for words

```
alphabet  = {0, 1, 0, ..., 0, 0, 0, ..., 0,0}
car       = {0, 0, 0, ..., 1, 0, 0, ..., 0,0}
corridor  = {0, 0, 0, ..., 0, 0, 1, ..., 0,0}
zebra     = {0, 0, 0, ..., 0, 0, 0, ..., 1,0}
```

At this point, it is obvious that with a very big corpus the dictionary of words can also be quite big. This means that if one would like to train a model using this corpus, one would need to provide a vector of a size at least with the number of words in the dictionary. Often more words are required for the input. If a trained model is for example a neural network model, this results in very big network that tries to make sense from many zeros and very few ones at its input and output layer. This is the perfect example of a sparse feature space.

The opposite of the sparse feature space is a dense feature space where embeddings come into help, especially word embeddings in context of NLP. Word embeddings are another representation of words in a corpus that assigns a C-dimensional vector of real values to each word, making it possible to distinguish one word from the other. Number C is a hyperparameter chosen for each corpus that is the best fit for it. It is much smaller than the size of the dictionary, but big enough to keep all relevant information about relations between words. If a dictionary for example has 50'000 words, embeddings may have the size of 100. It is far denser.

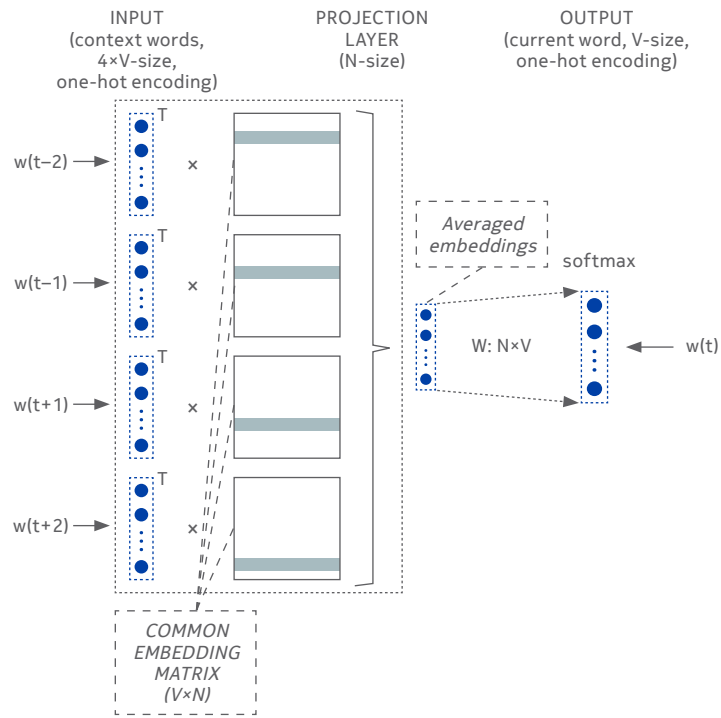
2. Very brief history of word embeddings

Word embeddings have their beginning in the 1960s [2] as a vector space model which is an idea to represent text documents, words, or any objects in general as vectors in vector space. A given vector, as mentioned in the previous section, can be quite big, so the reduction of dimensions was needed and which led to the latent semantic analysis (LSA) in the 1980s. The LSA [3] is an NLP technique used to analyze relationships between terms in documents. It assumes that words that are close in meaning will occur in similar pieces of text, which is the distributional hypothesis [4]. This was followed by the work done by Bengio et al. who provided a series of papers on reducing high dimensionality of word representations by "learning a distributed representation for words" [5]. This last paper proposed a feedforward neural network (neutral network language model (NNLM)) that was used to jointly learn the word vector representation and a statistical language model. The idea of learning word representation using a neural network was directly used in the word2vec [6] algorithm, which is one of the main algorithms that produce word embeddings. Another method – a statistical one, that has its roots in LSA – is Glove [7], which uses global statistics over an entire corpus to produce vector representations of words by keeping relations between them.

3. Algorithms

Both algorithms are unsupervised and are used to find a solution to the same task - create dense vector representations that not only put similar words close to each other, but also preserve multiple degrees of similarities and maximize accuracy of vector operations between words [8]. This means two things:

Figure 4 Detailed architecture of CBOW neural network model



A) Having a small error in vector coordinates might result in a different word but the overall meaning will remain the same or very similar.

B) It is possible to ask for a corresponding word with a different meaning like "what is the equivalent of 'uncle' having the following relation 'mother' <-> 'father'".

3.1 word2vec

It is a feedforward neural network, similar to the NNLM, but instead of learning embeddings and a statistical language model at the same time, it gets one or several (one-hot encoded) words as input and one word as output and, as a result, it learns only embeddings. It is a predictive model which means that it tries to predict a word from its context instead of calculating statistics over an entire corpus at once. There are two variants of this algorithm – continuous bag-of-words (CBOW) and skip-grams. Both are trained by going through the corpus with a sliding window which is created by a single word and its context words.

Figure 5 Example calculation from INPUT to PROJECTION layer

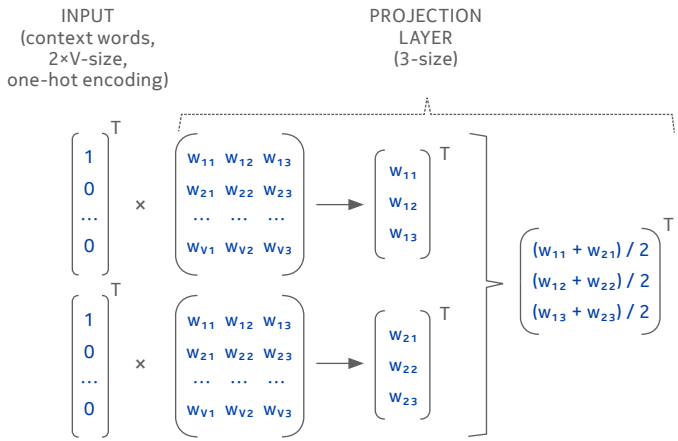
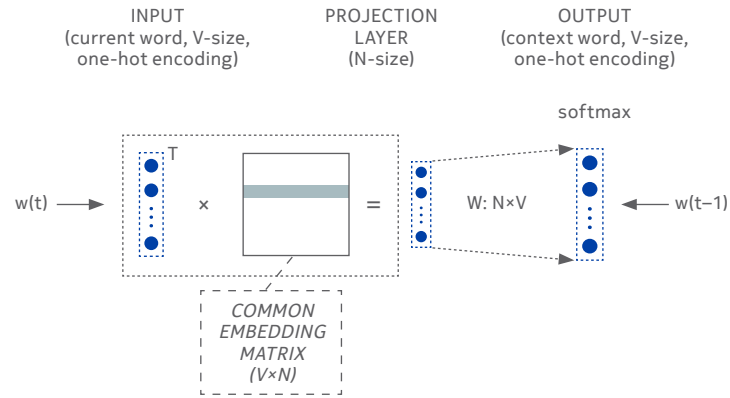


Figure 6 Detailed architecture of a skip-gram model for *w(t), w(t-1)*



The first variant, that is CBOW, is defined by the task of predicting a given word $w(t)$ by providing a few previous words $(w(t-1), w(t-2), ...)$ and a few future words $(w(t+1), w(t+2), ...)$. The architecture of this model is very simple, as there is no non-linear hidden layer and the projection layer is shared for all words. Therefore, vectors of input words are just averaged. The number of neurons in the projection layer defines how big is the embedding vector. A detailed architecture is presented in **Figure 4** and an example calculation in the projection layer in **Figure 5**. The name bag-of-words means that the order of words does not influence the projection. According to Mikolov et al. [6], the best results were achieved with four previous and four future words.

The second variant, that is a skip-gram, takes the opposite approach because instead of trying to predict the current word from the context, the task is to predict context words from current words. The maximum distance C from a current word is defined for such situation. This forms a window around the word from which output words are sampled. The more distant the word, the lower the weight in sampling. The number of words that is selected in learning is chosen by randomly selecting number R in the range $<1; C>$. R is the number of previous words and future words that are outputs, this gives $2 \times R$ words for a current word. To simplify the network instead of predicting all context words at once, each word is put one by one as a single output to the same input word. **Figure 6** shows a simplified skip-gram neural network for the following pair $w(t), w(t-1)$.

For both variants, the backpropagation algorithm is used to adjust and form final word embeddings.

3.2 GloVe

GloVe stands for Global Vectors [7]. It is an example of a count-based method. This algorithm at first step creates a word co-occurrence matrix X where each element of a matrix represents how often word i appears in context of word j. Each element is created by scanning the corpus in a way that for each term other term is checked within an area defined by the number of words (*window_size*) before the term and after the term. In addition, less weight is given for more distant words, using the following formula:

$$decay = 1/offset$$

Word vectors are defined with the following constraints equation:

$$w_i^T w_j + b_i + b_j = \log(X_{ij})$$

Where w_i is the vector for a main word, w_j is the vector for a context word, b_i and b_j are scalar biases for the main and context word.

To find out what are the word vectors, the following cost function is defined:

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

$f(X_{ij})$ is a weighting function which helps to prevent learning only from extremely common word pairs. The authors define the following function:

$$f(X_{ij}) = \begin{cases} (\frac{X_{ij}}{X_{MAX}})^{\alpha} & if\ X_{ij} < X_{MAX} \\ 1 & otherwise \end{cases}$$

X_{MAX} is a cutoff value that forbids a higher number of co-occurrences than a given value (the authors use $X_{MAX} = 100$ and $a = 3/4$).

Learning process optimizes the cost function by going over X matrix and adapting word vectors using AdaGrad algorithm (adaptive gradient descent, a modified version of stochastic gradient descent with a learning rate adaptation).

4. Benefits of using word embeddings

The first and the biggest benefit of using word embeddings in NLP is the one stated at the beginning of this article – it allows to use a dense hyperspace for words instead of a sparse one that contains mostly zeroes. This is not the only benefit, though. All described algorithms set the goal to create word vectors that preserve word similarities and both syntactic and semantic word analogies.

Syntactic word analogy can be checked by asking questions like: "'dance' is to 'dancing' as 'fly' it to '____'". Semantic word analogy, on the other hand, asks questions like: "'Athens' is to 'Greece' as 'Berlin' is to '____'".

Both types of questions represent the same type of sentence - "a is to b as c is to d" which can be written as:

$$w_b - w_a = w_d - w_c$$

By giving three words, the fourth can be found by calculating it through this equation and searching for the closest one using cosine similarity [8]. This is how this famous equation works.

$$\text{KING} - \text{MAN} + \text{WOMAN} = \text{QUEEN}$$

The similarity of words can be checked by searching for the closest words in a word vector hyperspace using cosine similarity on compared vectors. Words that can be used in the same context will be appear close to each other. This fact has also another consequence. Since analogical words are within similar distances to each other, it is possible to use clustering to form groups of similar words.

5. Conclusions

Word embeddings is a powerful NLP technique that allows to catch very useful properties of words in a dense word vector hyperspace. Its main uses can be found in higher level tasks that analyze sequences of words or generate them, for example machine translation. Using word embeddings is not constrained only to NLP. Words can be substituted with events, and by giving big sequences of events, it might be possible to find hidden relations that are not obvious.

References

[1] [Online]. Available: https://en.wikipedia.org/wiki/Text_corpus.

[2] "A brief history of word embeddings (and some clarifications)," [Online]. Available: https://www.linkedin.com/pulse/brief-history-word-embeddings-some-clarifications-magnus-sahlgren/.

[3] T. K. F. P. W. &. L. D. Landauer, "Introduction to Latent Semantic Analysis," in *Discourse* , 1998, pp. 25, 259–284.

[4] Z. S. Harris, "Distributional Structure," *WORD,* vol. 10, no. 2–3, pp. 146–162, 1954.

[5] B. Y., S. H., S. JS., M. F. and G. JL., "A Neural Probabilistic Language Model," *Journal of Machine Learning Research,* vol. 3, no. 3/1/2003, pp. 1137–1155, 2003.

[6] T. Mikolov, K. Chen, G. Corrado and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *CoRR,* vol. abs/1301.3781, 2013.

[7] J. Pennington, R. Socher and C. D. Manning, "Glove: Global Vectors for Word Representation.," in *EMNLP*, 2014, pp. 1532–1543.

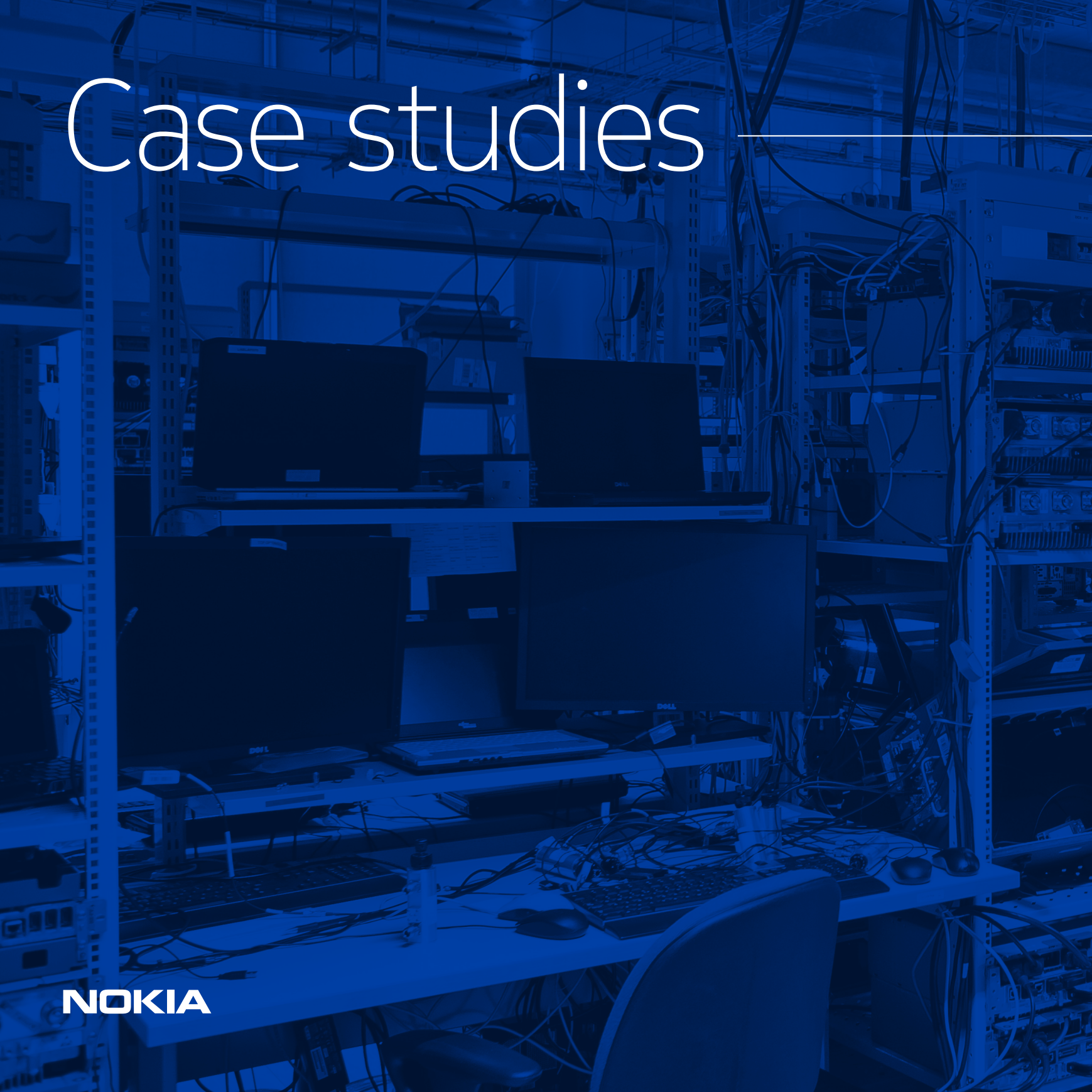
[8] T. Mikolov, W.-t. Yih and G. Zweig, "Linguistic Regularities in Continuous Space Word Representations.," in *HLT-NAACL*, 2013, pp. 746–751.

About the author

I am a graduate of Computer Science and Management faculty of Wroclaw University of Science and Technology with specialization in Intelligent Information Systems. I work as a Technical Lead Engineer in RAIN project and I am also involved in tasks related to machine learning. I am interested in a wide area of artificial intelligence, including applied machine learning, computer vision, and natural language processing.

Kamil Szatkowski
Technical Lead Engineer
MN BOAM TOOLS / Rain

Case studies



NOKIA

3.1

Maciej Norberciak
Birds, Bees and Evolved Antennas: Metaheuristic Methods and their Applications in Telecommunication

3.2

Dominik Deja, Aneta Stal
Contract Digitalization

3.3

Ireneusz Jabłoński
Financials monitoring and forecast with the Monte Carlo simulations

3.4

Jakub Kozerski
How we predict possible software defects during code review

3.5

Weronika Białecka
Recommender systems: introduction with use-case idea

3.6

Ewa Boryczka
Anomaly Detection: application in telecommunication networks

Birds, Bees and Evolved Antennas: Metaheuristic Methods and their Applications in Telecommunication

Maciej Norberciak
Senior Architect, Baseband Software
MN BB Baseband Architecture and Hardware Development

NOKIA

The number of devices connected to the Internet already surpassed the number of humans alive and is rapidly growing. Some forecasts even claim that there will be 125 billion (sic!) connected devices in 2030 [1]. In January 2017, the total population of planet Earth was around 7.5 billion people – half of them were using the Internet, two thirds were using mobile phones [2]. Cisco VN Index reports an 18-fold increase in mobile data traffic between 2011 and 2016 [3]. Telecommunications is a cornerstone of our modern society, the market that knows only one rule: constant, rapid expansion. However, to achieve financial viability constant optimizations must be carried out.

The aim of this paper is to introduce the reader to metaheuristics as a general-purpose optimization tool and to show some of their successful real-world applications in the telecommunication industry. A metaheuristic-aided antenna design, then antenna placement in various networks, finally a cellular network design and optimization are presented as examples. The article does not have an ambition of being a complete state of the art review, but lists selected bibliographical references that are easy to access and are written in a comprehensible way, without going into details on the methods used therein.

1. Metaheuristic methods – overview

Many of real-life tasks can be represented as optimization problems, hence developing efficient optimization methods is an intensively developing research area. Some of those tasks are considered "hard". For continuous optimization problems, it means that there is no known algorithm which allows finding a global optimum in a finite number of steps. For discrete optimization, it simply means the problem is NP-hard. For such tasks metaheuristic methods (which, in contrast to heuristics, are general-purpose, not domain-specific) yield very good results [4].

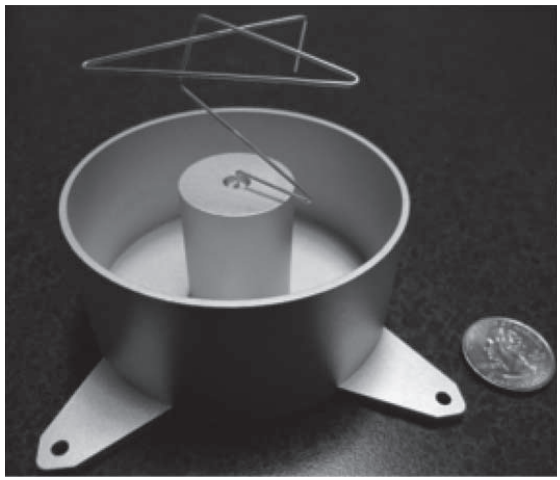
All the metaheuristics act randomly to a certain degree. This allows them to counteract the effects of combinatorial explosion – rapid growth of the problem's complexity caused by how the combinatorics of the problem is affected by its input and constraints. The trade-off is that the solution found is not necessarily the global optimum. Apart from few local search techniques (for example tabu search [5]) most of the metaheuristics are inspired by nature. First major source of inspiration is physics (or chemistry) – examples are gravitational search algorithm (inspired by the Newtonian laws of gravity and motion [6]), river formation dynamics (based on a manner in which water forms a river bed, taking into account soil erosion and sediment deposition [7]), harmony search (inspired by jazz musicians improvising in perfect harmony [8]), or simulated annealing (based on a metallurgical process where metal is heated up and then cooled in a controlled manner [9]). The second source of inspiration is biology, where one can find methods based on our natural evolution – various genetic and evolutionary algorithms (mimicking reproduction, mutation and recombination of genes, and natural selection [10]). Finally, swarm intelligence-based methods take inspiration from swarms or colonies of living organisms, for example ant colony optimization (pheromone-based communication of ants is mimicked [11]), artificial bee colony (simulating an intelligent foraging behavior of a honey bee swarm [12]), or firefly algorithm (inspired by flashing and a sexual attraction mechanism of Lampyridae insects [13]). As metaheuristics emulate nature's approach to problem solving and learn lessons from it on a more abstract level, they are all treated as a form of artificial intelligence.

2. Evolved antennas

On April 18, 2014, NASA's Lunar Atmosphere and Dust Environment Explorer (LADEE) crashed near the eastern rim of Sundman V crater on the far side of the Moon [14]. An eight-month mission was ended deliberately after the spacecraft run out of fuel needed to maintain its orbit. But the violent manner of LADEE's retirement was not unique – after all, humanity has purposefully struck moon with various devices since Ranger 7 sent first close up images of lunar surface back in 1964 [15]. What was unique was the antenna system that LADEE used to communicate with the Earth.

Designing an antenna is not an easy task. Just like in many other engineering problems, requirements and constraints are numerous and they interact with one another in a complex manner, and just like in many other engineering problems, concessions and tradeoffs must be made. The challenge of designing an antenna for a spacecraft is even greater, because factors like volume, mass, or power budget became very important. Finally, a delicate instrument must survive heavy Gs during lift-off and a somewhat difficult environment in space. The design process, despite numerous advancements in the field, remains largely manual, strongly relying on the engineer's experience and domain knowledge. The use of evolutionary metaheuristics in antenna design and optimization techniques has been investigated since the 1990s, and were triggered, not surprisingly, with a military application – using GA to design radar absorbing coatings [16]. As the field grew, computer speed increased and electromagnetic simulators improved, NASA has started to develop automatic antenna synthesis tools. First evolved antennas (backed up by a "traditionally" designed ones) went through their baptism of fire during Space Technology 5 (ST5) mission [17], showing better performance in every aspect while requiring less work to create [18]. Techniques developed for ST5 were perfected during LADEE design phase [19]. This time no backups were deemed necessary.

Figure 1 LADEE S-band omnidirectional evolved antenna (choke ring is 7.9 cm in diameter, antenna is 8.3 cm tall [19])

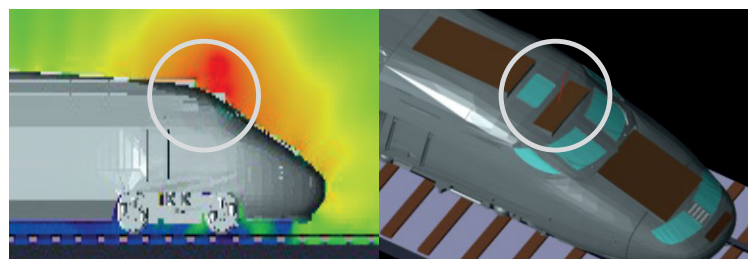


Naturally, space industry is not the only one interested in automatization of an antenna design process. Patch antennas, that is low profile devices that can be mounted on flat surfaces, have plethora of applications, for example, they are commonly used in mobile phones. Metaheuristics have been successfully applied to help with their design, for example in the form of evolutionary algorithms [21], ant colony optimization [22], or invasive weed optimization [23] (inspired from colonizing weeds which attack a field by means of dispersal and occupy opportunity spaces between the crops; weeds grow and reproduce individually and those which take more unused resources grow faster and produce more offspring). Other types of antennas, for example linear or elliptical arrays, are also being designed with the help of metaheuristics like cuckoo search [24] (based on a parasitic brooding behavior of some cuckoo species) or particle swarm optimization [25] (inspired by social behavior of bird flocking or fish schooling [26]).

3. Antenna placement problem

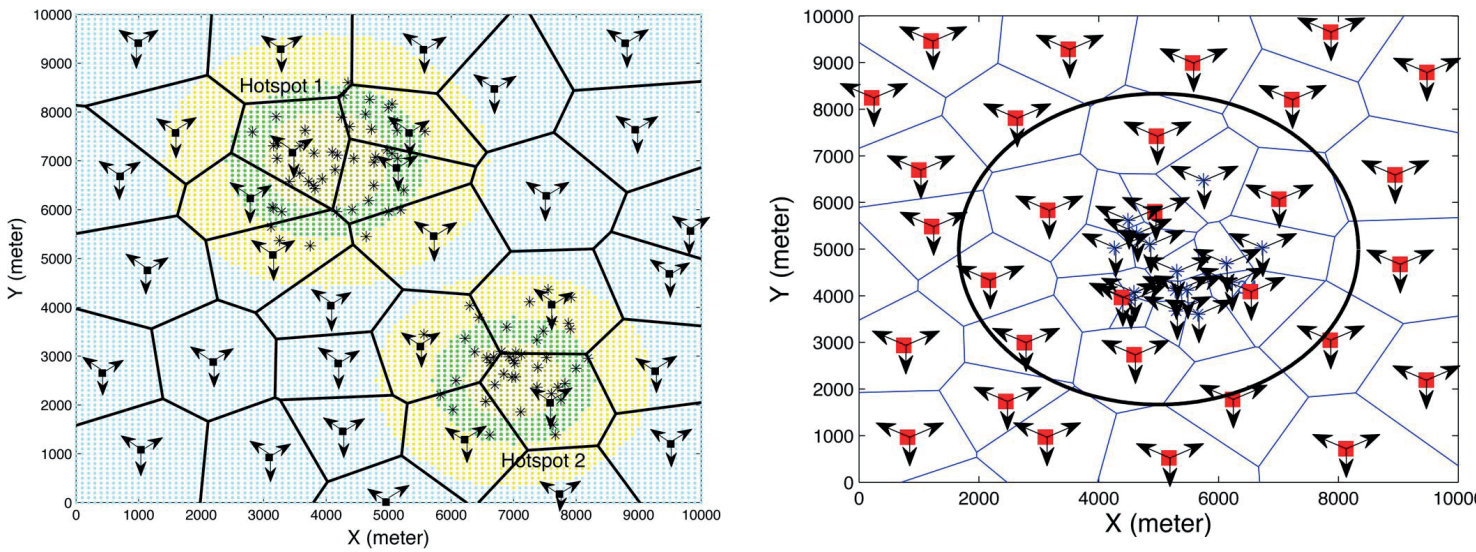
Advances in computing technology and fabrication techniques are expected to allow site-specific and evolving antennas, that are considering the electromagnetic characteristic of the environment in which the antenna is going to be installed. It was already successful, but as for now there is no reported research in areas other than satellite design [27]. In more "down to Earth" applications, commercially available and off-the-shelf products are typically used, that is, the design is not needed. Still, the problem of antenna placement on a platform (for example, devices, vehicles, masts, buildings) remains. Typically, this is done using engineer's expertise and judgement, but complex environment and interactions between an antenna and a platform may defy human's intuition when it comes to choosing an optimal antenna location. Combination of a powerful electromagnetics simulator and a robust optimization algorithm (for example metaheuristics) is needed to automate this task. Nowadays, almost all vehicles, for example trains, ships, airplanes, or cars, are equipped with antennas, which are ranging from "traditional" AM/FM, through satellite navigation, TV, to mobile network ones. Electromagnetic considerations must go along with the aerodynamic and sometimes even the aesthetic ones. Traditionally, antennas are placed on top of transportation platforms (the roof), but the optimal placement can be counterintuitive. First practical application of metaheuristics for a vehicular antenna placement was a receiving antenna on a train. The transmitter is located along the tracks and the goal is to receive the strongest signal possible. A genetic algorithm has found that the antenna should actually be placed between the roof and the front windows. What is interesting is that all the top locations were discarded early on by the seventh iteration of the algorithm [28].

Figure 2 Optimal antenna location found by a genetic algorithm on the train



There is one serious disadvantage to roof-mounted antennas – they protrude from the vehicle. Engineers and designers try to make antennas smaller, so their profile does not affect aerodynamics or aesthetic properties of the vehicle. This challenge may be overcome with glass mounted antennas, built from thin conducting materials and laminated between layers of glass in the vehicle windows. An additional difficulty is to decrease the number of antennas – combine different aperture requirements into one physical antenna system. Finally, effects of wind noise, reflecting, and scattering vary from vehicle to vehicle, forcing the engineers to rely on "gut feelings" and "rules of thumb", and make multiple experiments to modify antenna configuration. Fortunately, both design and placement of the glass-mounted antenna can be done with the help of metaheuristics. A genetic algorithm has been shown to be an effective tool [29] that helps in creating combined LTE/GPS antennas and mounting them in the windshield.

Figure 3 Optimized LTE cell planning in a heterogenous network (left; macrocell eNB - square; small-cell eNB - star) and "green" planning, using less energy during the night (right; night eNBs - red, additional day eNBs - blue); arrows represent sectors [40]



Most mounting platforms are not designed to move, but that does not make the antenna placement much easier. One of the key concepts in the IoT is ambient intelligence, that is an environment which is sensitive and responsive to the presence of human beings. The environment must be able to determine location of the user to offer context-aware services. Radio frequency identification technology (RFID) is commonly used to achieve this goal in smart buildings – high precision must go hand in hand with low cost (which means a minimum number of antennas). Even such, one could think, trivial task, as positioning RFID reader antennas at the portals (for example doors or gates) is often too complex for classical optimization algorithms, and metaheuristics (for example genetic algorithms) must be used [29]. Larger-scale problems encompass whole, topologically diverse, floors of buildings. There differential evolution [31], genetic [32] and memetic algorithms [32] have been successfully used to plan sensor and RFID reader networks.

4. Cellular network design and optimization

Solving an antenna placement (sometimes called base station placement or cell planning) problem is one the first steps in the design of mobile telecommunications network. The goal of the latter is to find the best set of antenna locations, maximizing coverage and quality of service, while minimizing the number of antennas. There is a substantial number of variables and constraints that need to be taken into consideration. Antenna parameters (such as position in 3D, elevation, azimuth, and power) have to be determined, taking into account terrain features (for example high buildings, hills, or mountains), infrastructure (site access, electrical and core network connections), electromagnetic pollution, safety regulations and finally the cost (not only the antenna device itself, but also the lease for mast or rooftop space, for example Manhattan has over 1000 buildings with rooftop cell phone antennas, with a monthly fee up to 10000$ per site [34]). A manual design process is challenging, time-consuming, and error-prone. Therefore, automatic aid in form of metaheuristic-based optimization tools quickly attracted the attention of both researchers and mobile network operators. Genetic algorithms have been successfully applied to optimize antenna positioning in 2G [35], 3G [36], and 4G [36] networks. Other methods include particle swarm optimization [40] and a grey wolf optimizer [41] (inspired from a wolves hunting process) which have been used for LTE cell planning.

Cell planning aims at solving multiple problems at the same time: equipment location, spectrum utilization, power allocation, traffic load balancing, and so on. Such multi-objective optimization of a cellular network has been studied since GSM was introduced, with good results obtained thanks to for example evolutionary [39] or bat algorithms [43] (inspired by the hunting behavior of bats, who search the prey with the help of echolocation, changing pulse emission rate, frequency, and loudness [44]). The real challenge came with 3G, when cellular networks became heterogenous and optimization had to be done across all Radio Access Technologies

(RATs) simultaneously. A wide range of metaheuristics had been tested on theoretical models [42]. Some success was achieved with a simulated annealing-based technique, developed in cooperation between Nokia and Deutsche Telekom [46], but the best results reported come from another metaheuristic – coral reef optimization [47]. This algorithm is based on the process of coral reefs' formation, simulating different phases of coral reproduction (sexual – external broadcast spawning and internal brooding, and asexual – budding and fragmentation), and fight for space in the reef, which ultimately renders an efficient algorithm for solving large-scale, highly constrained optimization problems. Applying it to optimization of network deployment across all the technologies (2G, 3G and 4G) in Spain lead to a potential reduction of total investment cost in range of 400 million euros, which constituted 15% of base cost [45].

5. Summary

In 1832 Paul Ludwig Schilling von Canstatt has created the first working switched-current electrical telegraph. After more than 180 years of constant growth and innovation, we are living in the age of information. Vast data networks span our connected world – but to design them effectively humans need tools more sophisticated than the engineer's "gut feeling". To solve large-scale, highly-constrained problems we turn to our best teacher – nature. It is expected that nature-inspired methods and other metaheuristics will play an important role in solving the big optimization challenges of real-life telecommunication industry problems.

References

[1] IHS Markit, "The Internet of Things: a movement, not a market," 24 10 2017. [Online]. Available: https://cdn.ihs.com/www/pdf/IoT_ebook.pdf. [Accessed 31 08 2018].

[2] World Economic Forum, "What happens in an internet minute in 2017?," 31 08 2017. [Online]. Available: https://www.weforum.org/agenda/2017/08/what-happens-in-an-internet-minute-in-2017. [Accessed 24 08 2018].

[3] Cisco, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper," 07 02 2017. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html. [Accessed 24 08 2018].

[4] J. e. a. Dreo, Metaheuristics for Hard Optimization, Springer-Verlag, 2006.

[5] M. Gendrau, "An Introduction to Tabu Search," in *Handbook of Metaheuristics*, Kluwer Academic Publishers, 2002.

[6] N. Sabri, M. Puteh and M. Mahmood, "An overview of Gravitational Search Algorithm utilization in optimization problems," in *IEEE 3rd International Conference on System Engineering and Technology*, 2013.

[7] P. Rabanal, I. Rodrigues and F. Rubio, "Using River Formation Dynamics to Design Heuristic Algorithms," in *International Conference on Unconventional Computation (in LNCS vol 4618)*, 2007.

[8] X.-S. Yang, "Harmony Search as a Metaheuristic Algorithm," *Studies in Computational Intelligence,* vol. 191, 2009.

[9] R. Rurenbar, "Simulated annealing algorithms: an overview," *IEEE Circuits and Devices Magazine,* vol. 5, no. 1, 1989.

[10] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer, 1996.

[11] M. Dorigo, V. Maniezzo and A. Colorni, "Ant System: Optimization by a Colony of Cooperating Agents," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 26, no. 1, 1996.

[12] D. Karaboga and B. B., "Artificial bee colony optimization algorithm for solving constrained optimization problems," *LNCS,* vol. 4529, 2007.

[13] X. Yang and X. He, "Firefly Algorithm: Recent Advances and Applications," *Int. J. Swarm Intelligence,* vol. 1, no. 1, 2013.

[14] National Air and Space Administration, "LADEE mission page," 2014. [Online]. [Accessed 12 08 2018].

[15] "National Space Sciences Data Center - Ranger 7," National Air and Space Administration, [Online]. Available: https://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=1964-041A. [Accessed 12 08 2018].

[16] E. Michielssen, J. Sajer, S. Ranjithan and R. Mittra, "Design of Lightweight, Broad-band Microwave Absorbers Using Genetic Algorithms," *IEEE Transactions on Microwave Theory & Techniques,* vol. 41, no. 6, 1993.

[17] National Air and Space Administration, "Space Technology 5," 2006. [Online]. [Accessed 12 08 2018].

[18] G. Hornby, A. Globus, D. Linden and J. Lohn, "Automated Antenna Design with Evolutionary Algorithms," in *Space 2006, AIAA Space Forum*, San Jose, California, 2006.

[19] J. Lohn, D. Linden, B. Belvins, T. Greenling and M. Allard, "Automated Synthesis of a Lunar Satellite Antenna System," *IEEE Transactions on Antennas and Propagation,* vol. 63, no. 4, 2015.

[20] F. Villegas, "Parallel Genetic-Algorithm Optimization of Shaped Beam Coverage Areas Using Planar 2-D Phased Arrays," *IEEE Transactions on Antennas and Propagation,* vol. 55, no. 6, 2007.

[21] M. Randall, A. Levis, A. Galehdar and D. Thiel, "Using Ant Colony Optimisation to Improve the Efficiency of Small Meander Line RFID Antennas," in *3rd IEEE International Conference on e-Science and Grid Computing*, 2007.

[22] F. Mohamadi Monavar, N. Komjani and P. Mousavi, "Application of Invasive Weed Optimization to Design a Broadband Patch Antenna With Symmetric Radiation Pattern," *IEEE Antennas and Wireless Propagation Letters ,* vol. 10, 2011.

[23] K. Abdul Rani and F. Malek, "Symmetric linear antenna array geometry synthesis using cuckoo search metaheuristic algorithm," in *The 17th Asia Pacific Conference on Communications*, 2011.

[24] R. Bera, D. Mandal, S. Ghoshal and R. Kar, "Optimal design of concentric elliptical array antenna for maximum side-lobe level reduction using particle swarm optimization with aging leader and challengers," in *International Conference on Communication and Signal Processing*, 2016.
[25] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *Proceedings of IEEE International Conference on Neural Networks*, 1995.
[26] D. Linden, "A system for evolving antennas in-situ," in *Proceedings Third NASA/DoD Workshop on Evolvable Hardware*, 2001.
[27] J. Infantolino, M. Barney and R. Haupt, "Optimal Position for an Antenna using a Genetic Algorithm," in *Applied Computational Electromagnetics Society Conference ACES 2010*, 2010.
[28] P. Sindhuja, Y. Kuwahara, K. Kumaki and Y. Hiramatsu, "A Design of Vehicular GPS and LTE Antenna Considering the Vehicular Body Effects," *Progress In Electromagnetics Research* , pp. 75–87, 2014.
[29] C. Lee, "Maxmimizing Read Accuracy by Using Genetic Algorithms to Locate RFID Reader Antennas at the Portals," *Journal of Software, vol. 5,* 2010.
[30] S. Liao, C. Chen, C. Chiu, M. Ho and T. W. B. Wysocki, "Optimal receiver antenna location in indoor environment using dynamic differential evloution and genetic algorithm," *EURASIP Journal on Wireless Communications and Networking,* no. 235, 2013.
[31] Y. Yang, Y. Wu, M. Xia and Z. Qin, "A RFID Network Planning Method Based on Genetic Algorithm," in *Proceedigs of the 6th International Conference on Networks Security, Wireless Communication and Trusted Computing, Vol. 1*, 2009.
[32] T. Garcia-Valverde, A. Garcia-Sola, J. Botia and A. Gomes-Skarmeta, "Automatic Design of an Indoor User Location Infrastructure Using a Memetic Multiobjective Approach," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 42, no. 5, 2012.
[33] K. Pickert, "Invisible Wires," New York Magazine, 04 10 2004. [Online]. [Accessed 12 08 2018].
[34] F. Rofii, M. Toscanni and D. Siswanto, "Optimization of coverage and the number of base transceiver station towers using fuzzy C-Means and genetic algorithm," *Journal of Theoretical and Applied Information Technology,* vol. 93, no. 1, 2016.
[35] J. Munyaneza, A. Kurien and B. Van Wyk, "Optimization of Antenna Placement in 3G Networks using Genetic Algorithm," in *3rd International Conference on Broadband Communications, Information Technology & Biomedical Applications*, 2008.
[36] S. Lee, S. Lee, K. Kim, D. Griffith and N. Golmie, "Optimal Deployment of Pico Base Stations in LTE-A Heterogenous Networks," *Computer Networks,* vol. 72, 2014.
[37] H. Ghazzai, E. Yaacoub and M. Alouini, "Optimized LTE Cell Planning With Varying Spatial and Temporal User Densities," *IEEE Transactions on Vehicular Technology,* vol. 65, no. 3, 2016.
[38] M. Komala, I. Wahidah and Isikmal, "LTE Networks BTS Location Optimation with Double Step Grey Wolf Optimizer," in *11th International Conference on Telecommunication Systems Services and Applications (TSSA)*, 2017.
[39] S. Cahon, E. Talbi and N. Melab, "A parallel and hybrid Multi-Objective Evolutionary algorithm applied to the design of cellular networks," in *IEEE Mediterranean Electrotechnical Conference*, 2006.
[40] S. Parija and P. Sahu, "A Metaheuristic Bat Inspired Technique for Cellular Network Optimization," in *2nd International Conference on Man and Machine Interfacing*, 2017.
[41] X. Yang, "A New Metaheuristic Bat-Inspired Algorithm," *Studies in Computational Intelligence,* vol. 284, 2010.
[42] S. Mendes, G. Molina, M. Vega-Rodriguez, Y. Gomez-Pulido, G. Miranda, C. Segura, E. Alba, P. Isasi, C. Leon and J. Sanchez-Perez, "Benchmarking a Wide Spectrum of Meaheuristic Techniques for the Radio Network Design Problem," *IEEE Transactions on Evolutionary Computation,* vol. 13, no. 5, 2009.
[43] L. Hu, I. Kovacs, P. Mogensen, O. Klein and W. Stormer, "Optimal New Site Deployment Algorithm for Heterogeneous Cellular Networks," in *IEEE Vehicular Technology Conference (VTC Fall)*, 2011.
[44] S. Salcedo-Sanz, J. Del Ser, I. Landa-Torres, S. Gil-López and J. Portilla-Figueras, "The Coral Reefs Optimization Algorithm: A Novel Metaheuristic for Efficiently Solving Optimization Problems," *The Scientific World Journal,* 2014.
[45] S. Salcedo-Sanz, J. P.-F. A. Sanchez-Garcia, S. Jimenez-Fernandez and A. Ahmadzadeh, "A coral-reef optimization algorithm for the optimal service distribution problem in mobile radio access networks," *Transactions on Emerging Telecommunications Technologies,* vol. 25, no. 11, 2014.

About the author

I received my MSc and PhD degrees in computer science from Wroclaw University of Science and Technology. I have published a dozen journal and conference papers. My area of scientific interest is artificial intelligence in general, and metaheuristics in particular. I have been working in the telecommunication industry for almost 15 years, as a developer, architect, and project manager.

Maciej Norberciak
Senior Architect, Baseband Software
MN BB Baseband Architecture and Hardware Development

Contract Digitalization

Dominik Deja, PhD
Product Owner
NOPS CMBD CSBD Analytics Translation & Digital Transformation

Aneta Stal
Data Analyst
NOPS CMBD CSBD Data Management Team

NOKIA

1. Introduction

One of the most popular Data Science quotes says that *data is the new oil*. No words can better describe the journey we have been on for the last six months. Oil is a resource which is hard to get, and even harder to process. But treated properly, it can turn into "black gold". The same case is for data – it's hard to collect it, and it's even harder to process it. But once its power is unleashed, the impact it can provide to the business is tremendous.

The goal of this project was to provide a complete and working POC (*Proof-Of-Concept*) for a fully automated contract (or legal texts in general) data extraction AI engine allowing fast and reliable extraction of information useful for other Nokia projects. Those information, or, as they will be called in this article, *features,* may vary from simple ones such as *Effective Date* (a date when a contract enters into force), through harder ones (e.g. *Warranty Start Date* for multiple different product groups), up to the hardest ones, that is features which are fully custom made, and depend on multiple other features, e.g. *Low Risk on Liquidated Damages Clause*.

Until now, all the work in this field has been performed by employees manually. Because it's extremely time consuming, complicated, and tedious job, the company was looking for a way how to release the employees from duty, and assign them to more interesting, and value creating tasks, so that their talent can be fully leveraged. Also, as the work is done by humans, human errors and flaws are significantly impacting this process. Right now, it can be characterized as follows:

- Very slow extraction time
- Linear (with the respect to the number of contracts analyzed) cost
- Prone to human error (our estimate is that depending on training time of an employee, and the feature complexity, the average human error is 5–10%)
- Every new feature requires dedicated training for the employee

All those inefficiencies can be solved with machine help. It will help employees in two ways:

- By extracting automatically all the common information pieces
- By collecting contract data and allowing users to compare data collected from new contracts with wide range of statistics aggregated from earlier contracts
- By speeding-up the search process, so that users can find information they are looking for faster, and with greater accuracy

Thus, the goal was to construct an AI engine which would:

- Work faster than a human
- Be a one-time cost (while building and setting it to work)
- At the beginning, work with accuracy at least comparable to a human
- Autonomously increase its accuracy with time, so that it achieves super human accuracy when fully trained
- Provide useful text suggestions in case it's not able to identify the correct answer
- Learn how to extract new features effectively, and at low cost (provided there is sufficient, and correct learning input)

It's not possible to provide a full answer on how we approached and solved every aspect of this project, and many of components are still under development. Therefore, in this article we would like to focus on those parts, which we found the most interesting and for which we were able to deliver solutions with a solid amount of creativity and novelty. Also, as this project is very R&D in its nature, we will be very honest about potential limitations, opportunities, future work, and what can and cannot be done.

Also, we would like to thank Vincenzo Pinto and Katarzyna Bednarz for their support, Claudia Sikuciński for contracting related input and support, and Joanna Iczakowska, Monika Kinal, Karolina Niedziela, Dawid Parzyk, Sylwia Pawłowska, Oleksandra Pyvovarova, Maja Rogal, Aleksandra Rusin, Dominika Tumialis, and Karolina Woch for their dedication, hard work and positive attitude.

2. Methods

2.1 Overview of the problem

Since we had to start from a scratch, instead of jumping to what commonly is associated with Data Science, that is predictive modeling, we had to start with basics.

First, we had to define the business value of our project and identify what possibilities will be open due to the leverage this project will provide. Also, because contracts are among the company's most secret documents, we had to check what can and what can't be done – who owns them, where they are and can be stored, how they can be processed, and what are the authority limitations. While it sounds straightforward, finding all that information and clearing the path ahead was, and still is one of the most time-consuming parts of this project.

Then, since models require training data, we had to collect available contracts[1]. This required a lot of manual work, as we had to contact people responsible for storing the contracts, build the necessary

1 In fact, obtaining enough full contracts was not possible at the time of writing this article, so we had to rely on separate contractual documentation (e.g. single framework agreement with no additional schedules, annexes, additions, or changes).

trust, and organize the inflow of documents. In fact, due to the complexity, this is still an ongoing process. Yet, even having a contract was not enough – we still needed to label the data.

Labeling is one of the most resource demanding phase of many Data Science projects. It is required because just as it is not possible to teach a human how to read without giving them any text and training, one cannot expect machine to do so. Labeled samples are a machine equivalent of traditional human-used textbooks. And since modern techniques often require tons of data, finding a smart way of obtaining a high-quality labeled data is of the uttermost importance. Thus, we had to come up with an innovative way of labeling our samples. We developed a *Waves* approach, and it will be described in detail in section 2.2.1.

Having the dataset ready, we used state-of-the-art algorithms to automatize paragraph identification, and hard rules for value extraction. In short, AI engine first identifies the area of the text with an answer, and then by using complex hard coded rules, extracts the final value.

Finally, results are shown, and future work will be discussed.

2.2 In the beginning – data!

Since one of the key objectives of the project was to train models to detect passages concerning a certain aspect of a contract, we were facing the inevitable issue of data scarcity. It's often the case that within the whole contract, which may consist of thousands of passages, only one piece is relevant. This created a huge imbalance, where less than 1%, and often less than 1‰ of cases are *true positive*[2]. This created two basic problems:

1. Finding enough cases for the minority class might be hard and enormously time consuming
2. Training algorithms on a biased dataset may result in a biased algorithm

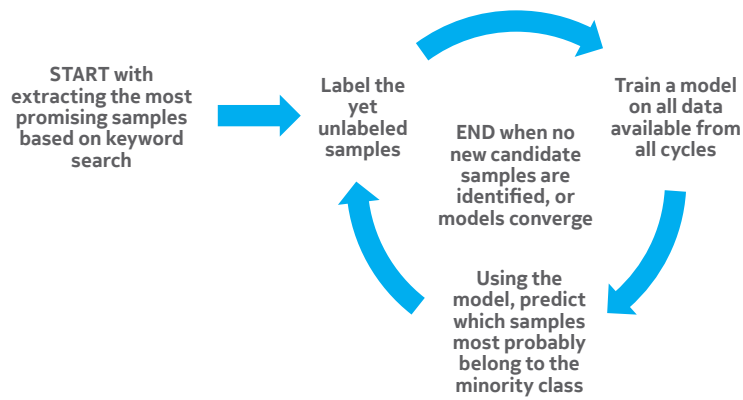
Both of those dangers were adequately addressed by *Waves* approach described in the following chapter.

2.2.1 Dataset creation – *Waves* approach

Let's say we would like to teach the machine how to recognize a piece of text which refers to *Effective Date*. Within the whole contract, which, on average, consists of 1000 text fragments, there's only one text fragment referring to this concept. Assuming that to train our algorithm we need at least 1000 examples[3], we would need to go through at least 1000 contracts and search for this piece. Since opening the file, and searching through it manually is complicated, given 10 minutes per contract[4], the whole search will take over 160 hours. And it's just for one, fairly simple, feature! For each new feature we would need to run this process all over again and for more complex features it may be even more time consuming.

Our approach to this problem was to start small, implement a human feedback loop, and gradually improve algorithms quality. By using it, we managed to reduce time needed to collect data for one feature from 160+ hours to about 40 hours[5]. **Figure 1** shows the simplified idea.

Figure 1 Waves approach for data collection



The process starts with our commercial contract support team trying to find the words which most probably will occur in the proximity of the searched feature. For the Effective Date such terms would be simply "effective date", or "comes into effect as of". But since we had to build a list of complicated logical conditions to sieve true positives from false ones, we called it a "decision tree". To find the right combination, for each feature, we tested several trees.

Based on trees, we selected samples which then were labeled by trained employees. Since texts we have analyzed came from OCRed pdfs, they were often erroneous, and split incorrectly. Thus, instead of labeling them as "yes/no" class, they had to use more classifications: 1 for those which clearly describes a given

2 For more information on statistical measures used in this article, it's worth to visit https://en.wikipedia.org/wiki/Confusion_matrix

3 As literature review suggested, with at least 1000 examples, there's enough data to predict whether a given feature can be possibly identified [3].

4 Based on our own experiences.

5 Assuming that with Waves one needs to check about 5000 text pieces, and checking one piece (in 95% of cases they are shorter than 64 words) takes 30 seconds, it takes approximately 42 hours

feature and contain an answer, -1 for those which clearly don't describe a given feature, 0 for those which are "in between", or describe a given feature but not fully, and 2 for those which describe a given feature, but instead of an answer, provide a trigger (list of conditions, or indicators to other parts of a contract).

Having the first batch of labeled samples, we trained the model. With trained model, we predicted, which unlabeled samples are most promising. With that selection, we asked trained employees to label the new sample. Then, having a new sample, we trained the model again, on an increased batch of labeled samples, coming from both decision tree, and first iteration of the model.

With such process going on for two, three, or more iterations, at some point all the data needed is labeled, and it's done in a significantly shorter time, than if we would go through one contract at a time. We estimated that the reduction of time is approximately four-fold, which means, that instead of providing one feature, we we'll be able to provide four, and this will enable us to provide more data at a lower cost (**Figure 2**).

2.3 Models training

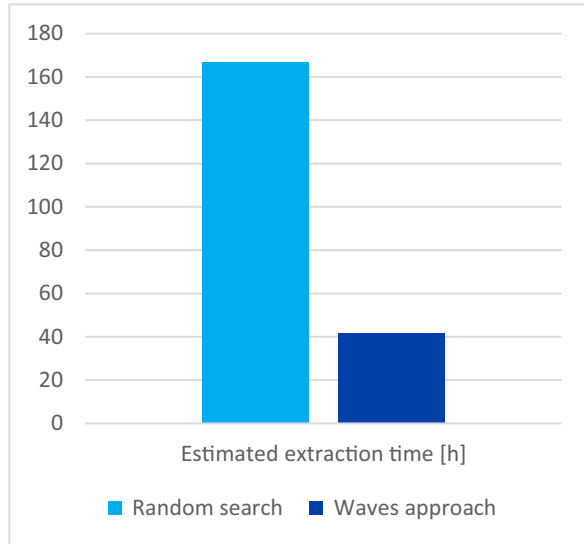
2.3.1 Word embedding

Since deep down in their "guts", computers are not capable of processing anything but zeroes and ones (or numbers in general), at some point, every text needs to be converted to a bunch of numbers. This is what we call a "word embedding".

There are several methods which can be used to obtain this effect. From one-hot encoding (a Boolean vector where the presence of a given word is indicated by value 1), frequency count (similar to the previous one, but instead of simple indication of presence, a number of occurrences is given), co-occurrence matrix (word by word matrix, where co-occurrence is counted – how frequently a word "A" occurs close to a word "B"), to more advanced methods such as global matrix factorization methods, local context window methods, or neural networks.

As all we had was approximately 1000 contractual documents of various quality (most of them being txt files coming from OCRed pdfs), we decided not to build a word embedding model ourselves, as it would be insufficient and severely damaged by OCR errors. Instead, we leveraged an open source pre-trained word vectors called GloVe (Global Vectors for Word Representation). GloVe was trained on 840 billion tokens (reduced later to 2.2 million most frequently used words) gathered from web data [1]. As a result, each word is represented by a vector of 300 numbers. Those numbers are then passed to the machine learning model.

Figure 2 Waves approach allows us to reduce time needed to collect the data for a single feature 4 folds



2.3.2 Machine Learning

To train an algorithm, the labeled data is needed. But how many labeled samples exactly do we need? There's no simple answer to this question. It depends on the problem complexity, algorithms used, data noisiness, the number of variables available and how strongly they relate to the analyzed phenomenon. While medical studies often need to provide answers with samples smaller than 30 cases, modern image recognition algorithms require hundreds of thousands of examples.

In our case, even though problems we are solving are complex, and data is very noisy and scarce, we were able to provide reasonably effective algorithms with training data being relatively small (<500 true positive cases).

As for the models, we found out that Gradient Boosting algorithms significantly outperform others, so we ended up using CatBoost model. CatBoost is an algorithm developed by Yandex, Russian equivalent of Google, and it's relatively new (first article about it comes from 2017, is still reviewed, and currently it's available via arXiv: [2]). As it's new, it solves most of the issues which were handicapping earlier models – it allows heterogenous types of data, works well with categorical data, it's robust and avoids overfitting, and outperforms most of the other algorithms with ease.

For each text piece, it provided us with a score, ranging from 0 to 1, which can be understood as a probability that a given piece

describes, or belongs, to a given feature. Having text fragments scores, we can select the most probable ones, and out of those, extract correct values.

2.3.3 Value extraction

For features, where a simple value can be treated as an answer, we added some hard-coded procedures enabling us to retrieve this information. For example, in most cases, the answer to the Effective Date is either a date, or a trigger (a list of conditions which must be met so that a contract can become binding). Since triggers are usually custom, and we don't possess an exhaustive list of all possibilities, we hard-code it in a way that the machine can recognize whether a final answer is a date (if so, then it returns a date), or a trigger (if so, then it returns the whole piece of text as an answer).

3. Initial Results

Initial results indicate that, at least for the simplest features, fully automated solution is possible[6]. We had to anonymize feature names, due to confidentiality reasons.

Table 1 Sample size after each wave (in brackets the difference between second and first wave is indicated)

	First wave		Second wave	
	1	-1	1	-1
Effective Date	240	1863	300 (+60)	3106 (+1243)
X_2	318	64	445 (+127)	3627 (+3563)
X_3	62	189	62 (+0)	189 (+0)
X_4	494	198	536 (+42)	522 (+324)
X_5	610	520	844 (+234)	1287 (+767)

What's probably most interesting is that the Waves approach works. Positive samples from the second wave often didn't have any of the keywords used to find samples in the first wave. This shows that even weak models exceed keyword search. Also, as it's shown in table 1, data is scarce, and it's very hard to find the actual true positives (checking the most likely text pieces recommended by model trained on first wave often resulted in minor increase of 1, and huge increase of -1 (e.g. for *Effective Date* it's +60 vs +1243).

6 When it comes to more complex ones, the research is ongoing.

Moreover, as it can be seen table 2, more heterogenous data is collected, it's harder for algorithms to correctly predict the outcome. It's because in the first wave, data is less complicated and comes from similar paragraphs of the contract (due to the decision tree search limitations), while in the second, models are able to search wider.

We can expect that within potential third wave, scores will slightly improve, but due to the small size of current dataset, it might happen that there's no more positive samples, so to further improve those scores, we need to step up and leverage more advanced models.

Table 2 Recall and Precision for each feature

	First wave		Second wave	
	Recall	Precision	Recall	Precision
Effective Date	78	70	83	32
X_2	89	81	78	76
X_3	95	83	95	83
X_4	89	88	91	86
X_5	69	80	93	84

4. Future work

Our overall, long term goal is to allow highly skilled, and qualified employees to focus on problems which will truly involve their creative capabilities, while the machine will do the rest.

While it will require many changes at the organizational level when it comes to the technical aspects of our solution, we plan to:

- Gather more labeled (in terms of both, number of features, and number of labeled samples per feature), and unlabeled (in terms of full contracts) data
- Develop further the AI engine so that it's able to work on full contracts (for contracts which lasted for a long time, it's often the case that conditions changed with time, and AI needs to be able to capture those changes)
- Include taxonomy of the most popular triggers (e.g. for *Effective Date* one of them would be "contract enters into force immediately after signing it by all involved parties")
- Develop AI engine so that it can capture more complex features, which rely on multiple clauses and conditions (e.g. *Effective Date* relying on *Signature Date* – whereas this one depends on only one additional feature, it's often hard to retrieve it anyway, as

signing date is often hand-written and OCR fails to recognize it – imagine how many difficulties there can be for more complex features!)

- Move from CatBoost to CNNs (Convolutional Neural Networks), since, with proper architecture, they scale up better, work faster, and allow to extract and use more information from the data
- Develop a user-friendly interface for employees to use this solution as their everyday work tool.

References

[1] J. P. a. R. S. a. C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Empirical Methods in Natural Language Processing*, 2014, pp. 1532--1543.

[2] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush and A. Gulin, "CatBoost: unbiased boosting with categorical features," arXiv preprint arXiv:1706.09516, 2017.

[3] G. V. a. M. R. G. Cormack, "Scalability of continuous active learning for reliable high-recall text classification," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2016.

About the authors

Dominik Deja obtained his PhD in Informatics at the Polish-Japanese Academy of Information Technology, where he focused on developing new Data Science methods. In Nokia, he's a Product Owner responsible for digital transformation – revolutionizing the company from the inside so that it can become the leading digital force at the global scale. As an ex-McKinsey Advanced Analytics consultant, experienced business speaker and university lecturer, he's always keen on sharing knowledge, so feel free to contact in case of any Data Science related topics.

Dominik Deja
Product Owner
NOPS CMBD CSBD Analytics Translation
& Digital Transformation

Aneta Stal graduated in Mathematics, Physics and Sinology from University of Warsaw. Shortly after finishing studies, she left Poland for Asia for 5 years and became a digital nomad doing Data Science remotely. From November 2017, she leads digitalization of archives of International Tracing Service in Germany. In April 2018 she fell in love in Wrocław, and she is currently working on contract digitalization in Nokia.

Aneta Stal
Data Analyst
NOPS CMBD CSBD Data Management Team

Financials monitoring and forecast with the Monte Carlo simulations

Ireneusz Jabłoński
Data Scientist
NOPS CMBD CSBD AI Lab

NOKIA

The purpose of this chapter is threefold: educational, research and informative. It operates in the field of data science, where the system and its behavior are perceived through the lens of randomness. Probabilistic mathematical modeling is first outlined and then applied in a case study of Monte Carlo simulations. Theoretical fundamentals are explored based on the application area of business financials monitoring and forecast, which to the readers can use to solve analogous problems of another nature (e.g. technical, biological, etc.). The potential for further development of the designed simulator as well as an enhancement of interpretations is discussed in the final section.

1. Introduction

Data science (DS) is a discipline designed to gather knowledge and information on system organization and its performance; here, the data is the carrier of knowledge and information on the system under study and the processes evolving within its boundaries. In this sense, data science is perceived by business users as a tool rather than a goal in and of itself. On the other hand, DS is normally perceived by researchers and scientists from the perspective of studying and potentially improving the tools and methods that are used to achieve real life objectives.

Many of the conceptions and techniques available in data science are associated with the science and business expectations, i.e. a user might be interested in learning about the structure and/or function of the system, studying its historical evolution, actual performance and/or prospective evolution over time (or in another domain of observations). One person can only be interested in describing the observed states, whereas others may require specification of prescription for emerging and/or evolving processes and properties. Finally, algorithms tend to be more reliable when applied to complex systems and data sets, in static and dynamic conditions, in off- and/or on-line mode, and more and more suitable for process automation. All this explains why data science techniques include data mining, big data analysis, data extraction and data retrieval. It also shows why data science concepts and processes are derived from data engineering, statistics, programming, social engineering, data warehousing, machine learning, and natural language processing, among others.

Nokia business is multidomain and represents technical, financial and social dimensions, which normally are considered separately. However, according to the well-known concept of synergetic gain, a holistic approach should be applied in order to obtain multidimensional information on the performance of the entire system, rather than of its component parts. The methods of artificial intelligence (AI), including machine learning, are significant players in the exploration of partial and integral business processes and properties, and can be applied to automated description, prediction and prescription tasks. Finding solution of a business problem normally meets the requirements provided by stakeholder, and the whole spectrum of data science techniques typically offers multiple algorithms possible for application for a given set of initial conditions (including the process, the data amount, and its quality).

In this chapter, a selected concept from the domain of monitoring and forecasting is presented as a basis for a case study of financial performance management. The role of budgeting and forecast in business monitoring is outlined together with Monte Carlo approach, which enables us to form conclusions about system evolution for the identified set of features and properties reconstructed from historical data.

The chapter can be used as an introduction to Monte Carlo simulations. It proves the significance of this method for business. In fact, although this approach is appropriate for mathematical statistics and not directly for the AI/ML methodologies [1-3], it still is an effective method of system and data exploration, and thus should be known in the business community of data analysts and data scientists. The solution outlined in this paper can be an interim step between time-series forecasting and detailed, structural physical-mathematical modeling.

2. Theory of probabilistic modeling in performance management

2.1 Performance management

A performance management framework consists of three main components: planning, budgeting and forecasting, where companies can seamlessly link top-down, strategic targets to financial and operational forecasts and report the performance against such targets.

The role of planning is to formulate a top-down strategic plan that defines the strategic aims of the enterprise and high-level activities required to achieve the goals of the organization.

Budgeting provides the assumption for a budget that enables resource allocation to be aligned to strategic goals and targets set across the entire company.

Modern business strongly relies on forecasting that tracks the expected performance of the business, so that timely decisions can be made to address shortfalls against target, or maximize emerging opportunity.

A fully integrated performance management framework is essential to provide corporate visibility of the activities that directly deliver

growth, and provide a clear framework for determining how to continuously allocate resources to support the strategy.

2.2 Probabilistic mathematical models

To analyze complex financial systems, apart from their deterministic (econometric or empirical) equivalents, models of probabilistic character are used. They depict the quantities changing randomly. A typical example of such approach is modeling of various noises as stochastic processes. It enables us, for instance, to perceive the noise contained in empirical data as the realization of a random variable. To be more precise: each recorded value of noise (i.e. a consecutive corrupted sample of the acquired signal) is interpreted as the realization of the following random variable. The most frequent assumption is that these variables are independent and identical, i.e. characterized by parameters of the same values (which makes the mathematical operations easier, among other things). These assumptions enable the application of common statistical rules. Nevertheless, one should remember that it is just a simplified description of the real phenomena, introduced to quantitatively estimate their contribution to the system's (e.g. electronic device's, business unit's) performance.

The random variable x is represented mathematically with its discrete probability distribution $p_x(x)$ (discrete variables) or with the probability density function $f_x(x)$ (continuous functions). Two of the most important parameters used to characterize random variables are the moment of the first order – expected value (the most probable value):

$$\mu_x = \mathrm{E}\{x\} = \int_{-\infty}^{\infty} x f_x(x) dx \qquad (1)$$

and the variance (the measure of dispersion of the realization around the expected value, equal to the square of the standard deviation):

$$\sigma_x^2 = \mathrm{Var}\{x\} = \mathrm{E}\{(x-\mu_x)^2\} = \int_{-\infty}^{\infty} (x-\mu_x)^2 f_x(x) dx. \qquad (2)$$

If the knowledge of particular distribution is required for calculations, then an adequate hypothesis should be assumed, and its statistical validation delivered. Typically, it is assumed that the variable has normal distribution (its realizations have white noise properties) [1]. The parameters of the random variable also are often unknown *a priori* and need to be calculated based on the recorded data. Evaluations of this kind are called the estimators, and its calculated values – the estimates. For N recorded realizations of the independent random variable, the expected value is estimated by calculating the mean value:

$$\hat{\mu}_x = \frac{1}{N} \sum_{i=1}^{N} X_i. \qquad (3)$$

An unbiased estimator of the variance is calculated as follows:

$$\hat{\sigma}_x^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \hat{\mu}_x)^2. \qquad (4)$$

It should be noted that the results of mathematical operations with random variables are also random variables.

2.3 The Monte Carlo method

Proposed by Stanisław Ulam, and developed further by John (János) von Neumann, the Monte Carlo method (MC) has multiple applications [2, 3]. For example, it is one of the ways for numerical integration. MC also enables performing statistical simulations, i.e. computer realizations of probabilistic mathematical models. In this application, the Monte Carlo method consists in generating random variables with modified distributions to estimate their parameters (typically, the expected value and the standard deviation).

In practice, the estimation of the random variable parameters $y = g(\mathbf{x})$ is the most significant part which is the function of the vector $\mathbf{x}$ of the other parameters with the known distribution f_x, or when $y = [x_1, x_2, \ldots, x_3]^{\mathrm{T}}$ is a result of a complex transformation of one variable. For example, the expected value of the considered variable y is equal to:

$$\mu_x = \mathrm{E}\{y\} = \int_{-\infty}^{\infty} y f_y(y) dy = \int_{\Re^M} g(\mathbf{x}) f_{\mathbf{x}}(\mathbf{x})\, d\mathbf{x}. \qquad (5)$$

It is calculated by transforming the integral with y into an integral with $\mathbf{x}$ and changing the region of integration into an M-dimensional space of $\Re^M$. Numerically, the parameters of the distribution can be estimated by automatically generating N samples of the $\mathbf{x}$ variable, and then transforming them according to the function $g(\mathbf{x})$ and calculating the result:

$$\begin{aligned} \hat{\mu}_y &= \frac{1}{N} \sum_{i=1}^{N} g(\mathbf{x}_i), \\ \hat{\sigma}_y^2 &= \frac{1}{N-1} \sum_{i=1}^{N} \left(g(\mathbf{x}_i) - \hat{\mu}_y\right)^2. \end{aligned} \qquad (6)$$

In some cases, the region of integration does not need to span the whole M-dimensional space, but its subspace Ω: $\Omega \subset \Re^M$. In this case, the following equation is obtained:

$$I_y = \int_{\Omega} g(\mathbf{x}) f_{\mathbf{x}}(\mathbf{x}) d\mathbf{x} = \int_{\Re^M} \mathbf{1}_{\Omega}(\mathbf{x})\, g(\mathbf{x}) f_{\mathbf{x}}(\mathbf{x}) d\mathbf{x}, \qquad (7)$$

where

$$\mathbf{1}_{\Omega}(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} \in \Omega, \\ 0, & \mathbf{x} \notin \Omega. \end{cases} \qquad (8)$$

There are also more advanced mathematical methods suitable for calculating integrals in Monte Carlo simulations [2, 3].

3. Business monitoring and forecast at Nokia

Nokia is a complex, multidomain company, which can be abstracted as a complex set of networks with numerous layers (technical, financial, social, etc.). Both the layers and their properties, represented within a given layer as vertices, are entangled in interconnections. One of the objectives of data science in this context is to disentangle these interconnections in order to infer the individual properties and links between them: historically, now, and in the future. This problem is not trivial and requires significant research effort, including sophisticated methodologies and tools, and this is why – especially for practical purposes – we simplify the system abstraction and/or select only a limited part of the system (and its behavior) for analysis.

The financial operation of Nokia business is the core of its existence and development, thus its temporal, socio-techno-economic conditions need to be understood in relation to the global and local tendencies. In fact, the company structure incorporates numerous teams responsible for strategic, financial, and operational processes. The Nokia AI Lab by CMBD is a team of data scientists skilled in systems and data exploration working on optimization *sensu largo* of Nokia business operation and management processes. One of the projects that the team has in its portfolio is monitoring and forecasting of selected Nokia Business Groups' (here: Mobile Networks – MN and Global Services – GS) financial performance based on the data created and accumulated in the LoA process. LoA (Limits of Authority) procedure incorporates actions directed toward preparation of the offer conditions for potential stakeholders. It refers to the technical content and financial details, and contract realization spanned in time after its signing. In the present case study, LoA Gate 4 (G4) data has been used (**Figure 1**). This data explains the lower limit (the worse) financial conditions for contract signing by Nokia. In reality, the process of contracting is complex since it includes 'Won', 'Ongoing', and 'Lost' deals for any point in time of business performance, each classified to one of the selected (A/A+, C, D, E) levels (according to concise criteria defined in Nokia), and whose technical and financial characteristics can be still reformulated at higher levels of the LoA procedure. For further details of the Nokia LoA process, see [4].

An exemplary problem formulated for data scientists can be expressed as follows: "Provide reliable projection and forecast of Nokia MN and GS (resulting) financials using G4 (information at offer creation level) data".

The solution of the problem has been limited to the exploration of only internal data, and is based on observation of the 'Net sales' (*NS*) and 'Sales Margin' (*SM*) characteristics ascribed to the corresponding deals (*NS* and *SM* definitions can be found elsewhere in [5]) and presented in the modes defined in the project (for more details see further chapters). Nevertheless, alternative approaches are possible, differing both with respect to the range of the data sets (internal, external) and the methodologies (in this project, the Monte Carlo simulation has been applied and described in the chapter, but other methods are possible). In our project, Monte Carlo methodologies provide quick and reliable insight into the evolution of the business as conditioned by the selected variables that were identified as the key contributors to the studied phenomena. In this way, we can obtain explanations of historical and prospective business developments as observed in practice.

3.1 Projection view

Projection is an explicit way of presenting Nokia's financial performance over the upcoming years. It employs the information generated for every single deal (Sales Opportunity (SO)) at the step of the case approval, that is, Gate 4. In the projection, cash flow (Revenue collection) of each SO is presented with respect to each quarter of SO's duration period. The projection contains the financial information from every sales opportunity, merged with respect to quarters and presented on a timeline (**Figure 2**). This is the basis for KPIs calculation, e.g. the temporal (year over year, quarter over quarter) change in *NS* and/or, *SM* the most impacting case(s), etc.

Figure 1 Selected aspects of the Nokia LoA process.

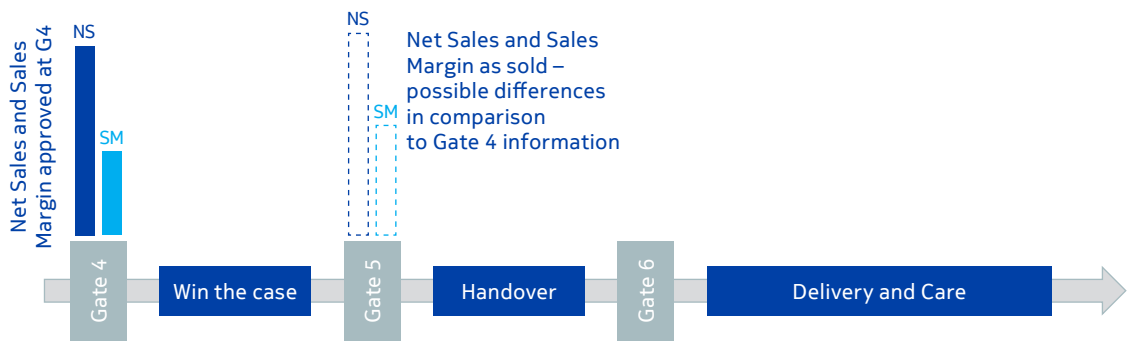
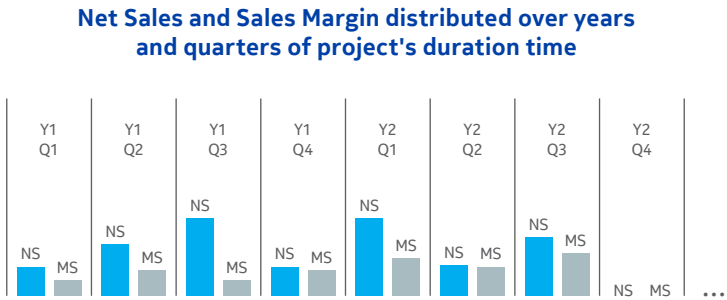
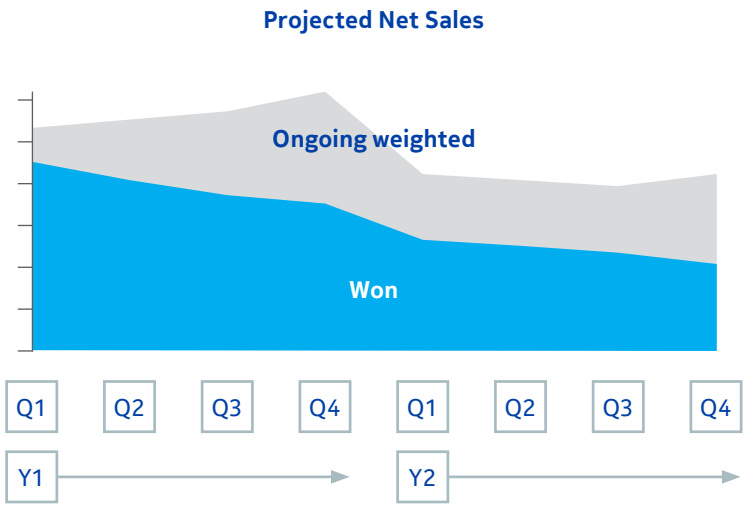


Figure 2 a) Single deal realizes uniquely in a timeline; b) The Projection view – presents the process of Net Sales collection (including SM level) for all sales opportunities we have the G4 record of, on a timeline with quarter resolution.

a)



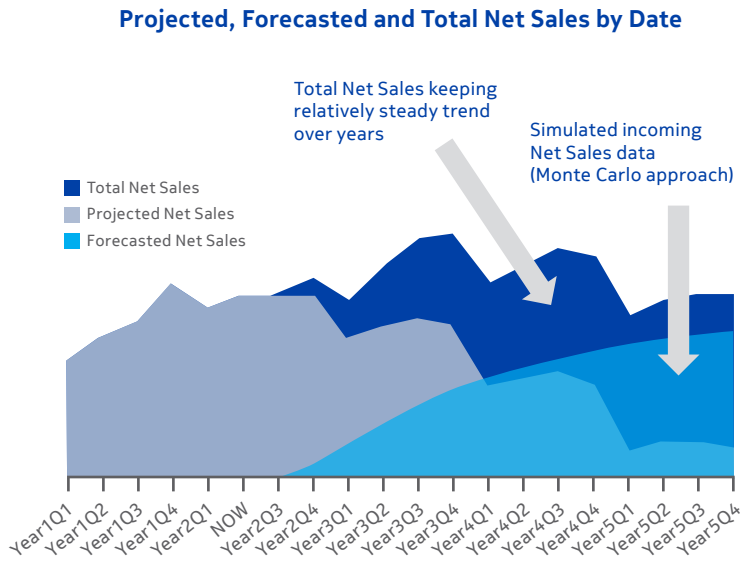
b)



3.2 Forecast view

The Gate 4 information only covers the present, but more Sales Opportunities are expected to be developed in future. Those future cases will be an important influence on Nokia's financial performance. This expected future data is generated by the forecast. It extends the run rate analysis by using the Monte Carlo technique, assuming that markets will behave similarly to the past. This way, the forecaster provides the information about the Nokia's future financial performance (**Figure 3**) based on its performance up-to-date.

Figure 3 The Forecast view presents the process of Net Sales collection (including SM level) for the sales opportunities we are expecting to have in the future, on a timeline with quarter resolution.



3.3 Some methodological remarks valid for the project

The financial data used to prepare the Net Sales and Sales Margin projections is extracted directly from the LoA data file. In the chapter, the information for individual MN and GS deals that was used starts in 2015, with monthly resolution. Calculations contain an appropriate aggregation of the *NS* and *SM* characteristics, expressed originally per deal. In this way, monitoring of projected financial characteristics is available at the Global, Business Group (BG), Business Line (BL), Market and Customer Team (CT) levels with quarter resolution.

The forecast provides the information necessary to form conclusions about the future financial situation of Nokia. It means that we need to reconstruct the temporal process of an incoming deal, together with its timeline contribution to the whole Business Group financial portfolio observed in the projection view. The Monte Carlo (MC) simulation was proposed in the project as a reliable methodology of Business Group financials forecast (here: MN and GS), relying on the assumption that both incoming deals and their properties can be perceived as random processes. A general schema of the MC algorithm used in this work is depicted in **Figure 4**.

Figure 4 General schema of the Monte Carlo procedure applied for the MN and GS financials forecast.

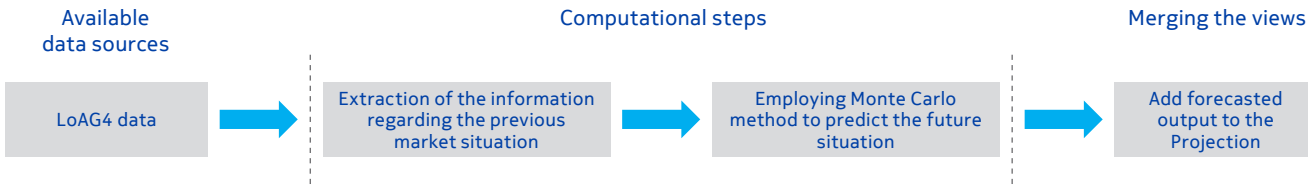
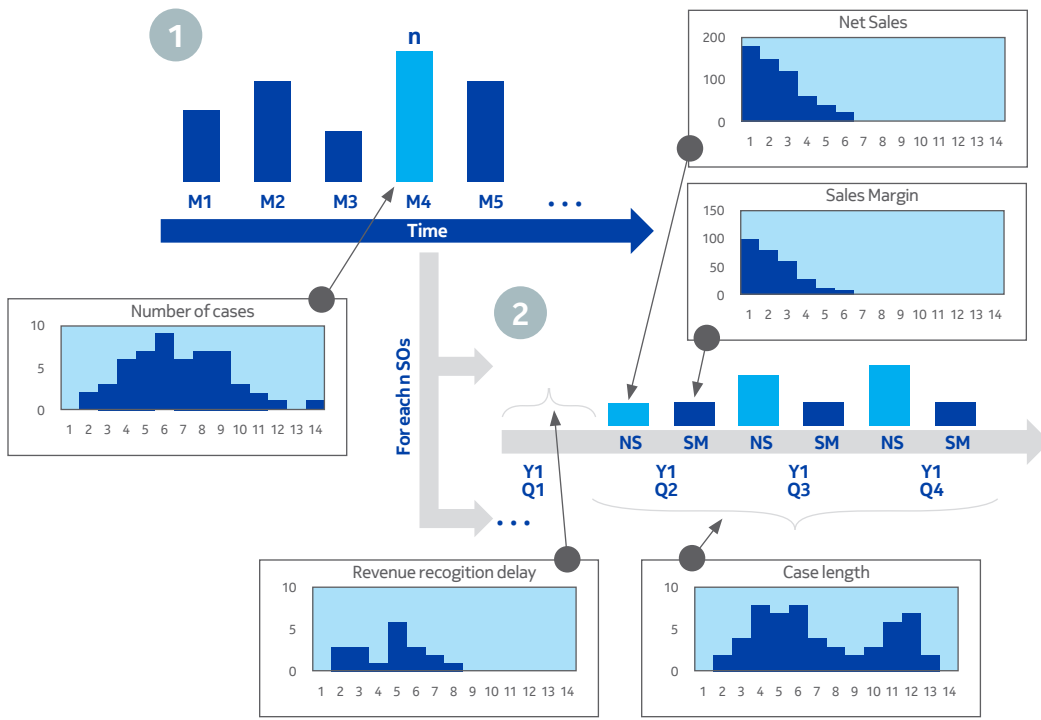


Figure 5 Probabilistic mathematical model of the MN and GS deals.

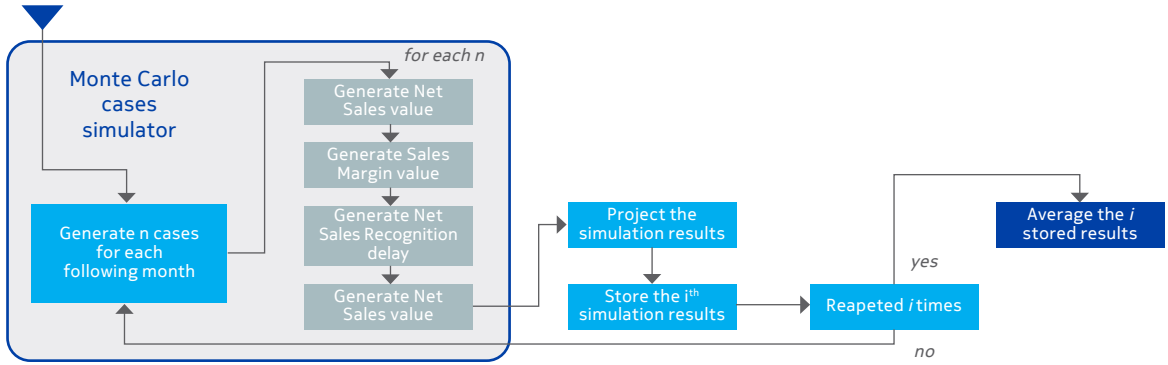


The strategy for expressing the future behavior of Nokia business should include reconstruction of these random characteristics from historical (here: LoA) data and their prolongation to the future. In the project, the experimental distributions for four parameters have been estimated, per each deal's level and for originally considered unit of time (month), i.e.: the number of incoming cases, net sales, sales margin, revenue recognition delay, case length. They can be used further in an iterative procedure of Monte Carlo simulation of prospective *NS* and *SM* characteristics.

Each run of the MC incoming case simulator consists of two major steps:

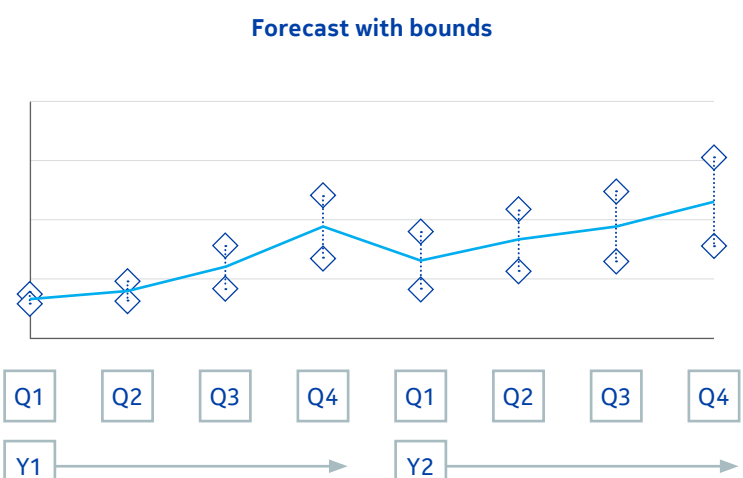
1. For each of the following months, a number of *n* cases is generated according to the reconstructed, experimental distribution.
2. For each of *n* cases, four KPIs are generated according to the proper distributions (**Figure 5**). Those four KPIs describe each case at a given level.

Figure 6 The flowchart for automated financial forecast.



Iteratively repeated, the MC case simulator generates the outlook on the prospective financial situation by creating synthetic samples of sales opportunities (**Figure 6**), which are likely to occur in the future. Results of each iteration are stored and averaged in the final step, to make the predictions less sensitive to random variation. Averaged results are passed to the projections and merged with them. Moreover, confidence bounds are calculated for the forecasted trends, which supplements the conclusions available for the users (**Figure 7**).

Figure 7 Forecasted trends (financials, e.g. Net Sales or Sales Margin) supplemented with the confidence bounds; in business, the lower bound can be interpreted as a pessimistic scenario, whereas the upper bound is understood as an optimistic scenario.

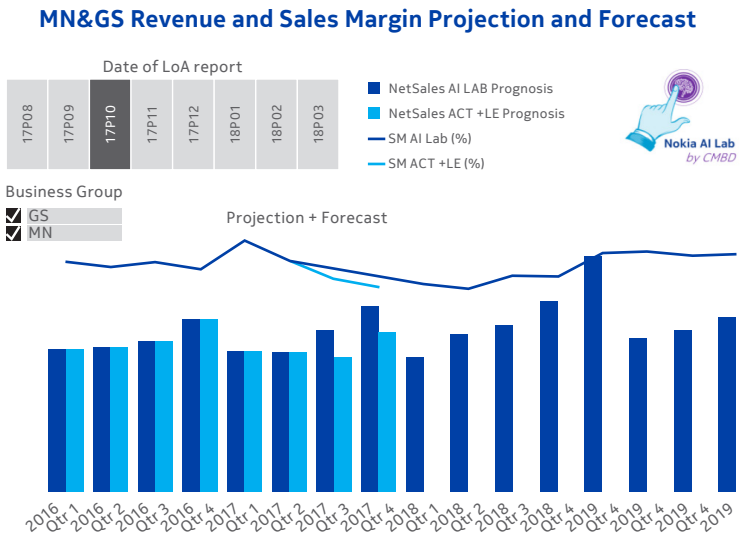


3.4 Results

Calculating Net Sales (*NS*) and Sales Margin (*SM*) projection and forecast provides an insight into the actual sales and financial status of Nokia business and their possible evolution in forthcoming quarters/years (up to five years). Since these views (projection, forecast) are based on G4 characteristics, thus further studies of their properties are required. The main question is whether G4 characteristics are good estimators for the materialized contract. The answer, in a graphical form, can be formulated based on the reference to the results of acting business (ACT) and latest estimates (LE) provided by financial units of Nokia. **Figure 8** demonstrates how the characteristics calculated as depicted in this chapter (marked as 'AI LAB') can be compared to the published results of company performance. From the analytical point of view, the systematic differences between the levels of the trends distinguished can be used to reduce the systematic error of indirect insight, i.e. changes between LoA G4 up to LoA G6 levels.

In its basic form, the projection mode provides the view of business performance (including prospective time horizon) as if we were to stop all offering activities today. The forecast mode broadens this regime of analysis with a dynamic contribution of deals able to generate income in the future. What is more, impact studies can be conducted for the proposed methodology, where the cases most impacting on observed financial output (*NS* and/or *SM*, positively and/or negatively) are identified and further studied in detail in order to provide recommendations and trigger actions (this is especially valid for ongoing deals and formulation of the business strategy for the company in the future). Finally, the designed and implemented tool is suitable for scenario simulation. For example, by applying changes in parameters distributions, the user can imitate more or less aggressive conditions on the telecommunications market, which may imply the erosion of price for the offered products.

Figure 8 Graphical demonstration of the projected sales financials against those forecasted with the Monte Carlo procedure; the data used to create this figure is synthetic (due to data compliance policy) and cannot be used in business practice.



4. Conclusions

The chapter discusses a selected aspect of data science where systems and processes are seen through the lens of randomness. Probabilistic mathematical description is still of great importance in business at any level of data exploration, i.e. data cleansing, data processing, and interpretation. System complexity to a large extent determines the adequate and entangled character of the observed output. Data science offers plenty of strategies for studying the properties and behavior of such a system, including e.g. reductionism and complexity, determinism, nonlinear dynamics, randomness, learning schemes etc. However, it is the data scientist/analyst who is finally responsible for the abstraction of the real problem and solving it according to the rules of these abstractions. This is why the knowledge, experience and intuition on the language of abstraction and the system under study is still an important trait of a contemporary data scientist, in spite of the availability of more and more universal procedures dedicated to studying systems and data sets.

Theoretical fundamentals of the probabilistic mathematical modeling and the Monte Carlo simulations were outlined in the first part of the chapter. Next, a typical problem of financial nature was formulated and solved, providing some practical remarks on the use of the Monte Carlo simulations for monitoring and forecasting the behavior of complex systems. The advantage of the presented methodology is its simplicity and availability for numerous analytical software products.

The practical example outlined in this chapter is of key importance for business performance management. Firstly, because the results of financial monitoring and forecast can be used by senior leadership team to make timely decisions that link top-down, strategic targets and assure their achievement. Secondly, the same results are the input for planning and budgeting. Finally, financial monitoring and forecasting make the dialogue between the company and its stakeholders more transparent and predictable.

References

[1] T. T. Soong, Fundamentals of probability and statistics for engineers, John Wiley & Sons Ltd., West Sussex, 2004.

[2] I. M. Sobol, A primer for the Monte Carlo method, CRC Press, Boca Raton 1994.

[3] R. Y. Rubinstein, D. P. Kroese, Simulation and the Monte Carlo method, John Willey & Sons Inc., Ne Jersey 2008.

[4] M. Kondrat, I. Jabłoński, ML controller for automation of contracts handling in the global telco company, Engineering Applications of Artificial Intelligence, 2018 (submitted).

[5] P. W. Farris, N. T. Bendle, P. E. Pfeifer, D. J. Reibstein, Marketing Metrics: The Definitive Guide to Measuring Marketing Performance, Pearson Education Upper Saddle River, New Jersey, 2010.

About the author

Post-graduate studies in Electronics from Wroclaw University of Science and Technology. Interested in research & engineering work on physical-mathematical modeling and experimental techniques. Currently focused in area of the complex systems measurement.

Ireneusz Jabłoński, Ph.D. D.Sc.
Data Scientist
NOPS CMBD CSBD AI Lab

Case Studies

How we predict possible software defects during code review

Jakub Kozerski
Engineer
MN BOAM Productization

NOKIA

Abstract

In software development, one of the most challenging goals is to create consistent and bug free code. To facilitate this, many different processes and tools have emerged. Due to breadth of the topic, the problem could be approached from different perspectives. For instance, one way to eliminate bugs or design defects is to use pair programming or the review process, but they are dependent on developers. Another common approach to ensure code consistency is using linters and other static code analyzers (SCAs). This family of tools generally gives good results, but the goal is to keep them warning free. Moreover, SCA tools are based on common knowledge and good practices in general, which in principle should be implemented inside coding guidelines. However, there is one more way to ensure code consistency, a somewhat uncovered area: namely, using custom made quality rules and considering the history of the project. The proposed solution consists of two parts. The first one is an ETL process, which regulates the maintenance of a database that contains information about the evolution of codebase. The second one is a system which marks potentially risky functions during code review, and then suggest solutions.

1. Introduction

Quality control in software development is a challenging topic. The main goal of this process is to create working and bug free software. In [1] Capers presents an estimated cost of a defect fix depending on the phase of development in which it was found. The cost of fixing a bug after release is approximately four times higher than during development and testing. Furthermore, fixing defects earlier can have positive effect on the overall software quality. S. Slaughter in [2] proposes two types of quality costs: conformance and nonconformance. The first type is divided into prevention (trainings) and assurance (inspections, testing) costs. Nonconformance costs in turn are about findings bugs, refactors and maintenance. To reduce the cost of quality without decreasing it, one should reduce failure costs to zero. A basic strategy known as the 'shift left principle' is to eliminate defects as early as possible. Assurance costs include code review, also called code inspection. In this article the author will focus on this part of cost optimization with simultaneous increase in quality.

During typical code review, developers submit changes to an appropriate code review tool (e.g. ReviewBoard, Gitlab, Github, Gerrit) and ask teammates to check their code. Reviewers can then validate the solution, suggest changes and improvements in order to increase code quality, and share knowledge. A good addition to this process are code linters and static code analyzers. For example, in Python it can be Pylint and for C++: CppCheck, clang-tidy, Valgrind, Sonar, Coverity, klockwork. These tools check for common mistakes and standardize code style, thus helping developers and reviewers to focus on important issues. Static code analyzers utilize only a set of rules based on well-known software development practices. In this article we propose an extension to these rules, which rules based on the history of the project. In fact, there are already similar projects, like Commit Guru [3] and Clever [4], that cover the whole pipeline from extracting data through creation of models to suggesting changes – but as of 2017 they weren't available as open source. Moreover, there is no appropriate ETL tool for extracting metrics from code. Therefore, the Veles was founded as a holistic solution to all those issues. Now it's a framework able to harvest data from version control systems like SVN or Git, store in an easy to use format, create models and integrate it all with review tools.

In section 2, we describe data extraction process and used metrics. Section 3 is about preparing datasets and creating machine learning models. In section 4 we cover the results and follow with conclusions in section 5. Section 6 presents ideas for other potential use cases.

2. Data

Preparing clean, correct datasets is very hard yet very important in ML. The standard procedure nowadays to harvest data is to create an ETL[1] pipeline, which should be automated and reproducible. Veles continuously monitors given repositories and for each new revision extracts data and stores it in an SQL database.

In the cited literature [3] [5] [6] [7] one can find many metrics that can be used for defect detection. Metrics can have different granularity, from file based, through class, to function or even scopes. In this article we focus on the function level. Features can be divided into two categories: those related to code and those based on the evolution of code over time. **Table 1** presents examples of code-related metrics. Veles utilizes existing tools, like Lizard [8] or Readability [9] to create metrics. The second category, that is, the evolution-related metrics, depends on revision metadata like *author*.

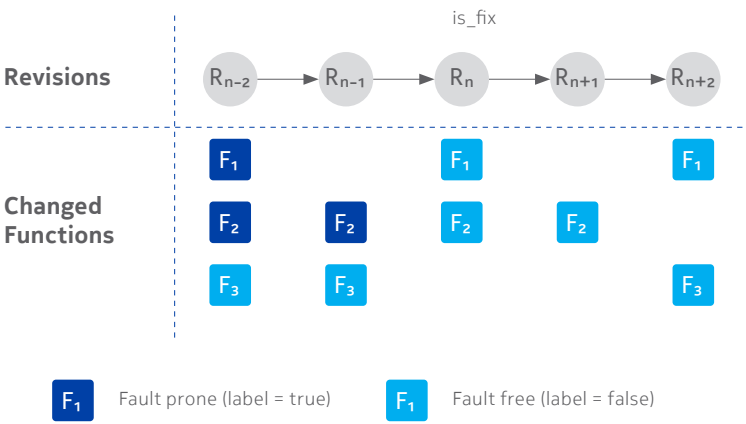
In **Figure 1** the circles at the top represent sequence of revisions. Each revision contains only changed functions represented as squares with rounded corners, and only these functions are measured. For example, the dataset created from **Figure 1** contains five revisions and three different functions spread to ten data points. Using such representation, Veles can track the evolution of functions over time, which allows for time-related metrics. A function is represented as pair: function name, file name. The drawback of this method is losing some data due to function overriding.

1 ETL - Extract, Transform and Load

Table 1 Example of code metrics

parameter_count	number of parameters (arguments) in a function or method
line_count	number of lines of code in a function or method
cyclomatic_complexity	McCabe cyclomatic complexity
author	person who changed the function
token_count	number of tokens in the function definition
flesch_reading_ease	Flesch readability score
days_since_modification	number of days since previous change
author_count	number of authors
revision_count	number of revisions
is_revert	true if function was reverted
is_fix	true if function was a bug fix
label	boolean value that indicates quality flaws, fault-proneness, ground truth

Figure 1 Example of label metric extraction.



To utilize supervised learning, the data should contain a information that represents ground truth or class. In other words, metrics can be split into predictor and predicted variable. In this data, a metric called *label* is used as the ground truth. The simplest solution is to set the label to *True* if any subsequent revision/commit changing the function contains a fix for bug, as presented in **Figure 1**. (There are also other techniques related to the nature of project or like SZZ [10] which are based on changed lines, but they are not used yet.)

Due to the nature of the problem, datasets can be unbalanced in terms of class distribution. This phenomenon can lead to overfitting or skewing the accuracy towards the majority class.

3. Model

Predicting software defects can be formulated as a binary classification problem. Here we chose the random forest algorithm [11] implemented as part of scikit-learn Python package [12].

Because the data is unbalanced, we propose to focus on f1-score instead of accuracy. Moreover, instead of classes, random forest can return a membership value. Using thresholding one can obtain the final classification. This method can trade recall reduction in exchange for higher precision.Machine learning models should solve specific problems. In the case of supporting code review, final predictions go to developers and reviewers. Therefore, we chose to favour precision by manipulating the threshold, so the precision is not less than 80%.

Training procedure is presented in **Listing 1**.

Listing 1 Training procedure.

```
# input: dataset
# output: classification model, scores
train_set, test_set = stratified_split(dataset) # 4:1

hyper_parameters = optimization(train_set, auc)
# optimization use average auc score obtained from
crossvalidation procedure

best_model, threshold = crossvalidation(train_set)
# best_model is retrived from models created during
crossvalidation precedure with related threshold value

scores = predict(best_model, test_set)
# final scores obtained on test_set
```

4. Results

Veles already works in production environment. Thanks to that the results presented in **Table 2** could be obtained from internal repositories. Additionally, initial surveys filled by developers, indicate that there is a need to provide additional explanations why a function belongs to fault-prone class.

Table 2 Results of experiments on real data.

	#1	#2	#3
Training set size	5027	15041	8743
# of fault-prone	2629	6209	2383
Unbalance of class	52%	41%	27%
Accuracy	0.82	0.72	0.83
Roc auc	0.91	0.79	0.86
Precision	0.80	0.80	0.80
Recall	0.87	0.44	0.54
F1-score	0.83	0.56	0.65

5. Conclusions

As presented machine learning consists of two parts. The first one is related to extracting, processing and cleaning the data, whereas the second one is concerned with modeling.

Generally speaking, the ETL part is much harder than modeling and our case was the same. In addition, this part was also highly time consuming. The relevant literature defines a lot of different metrics that can be used, but without discussing the appropriate tools or libraries. Therefore, in this study we explored only a fraction of them.

On the other hand, the modeling part was relatively simple. However, some studies suggest that using a simple approach can lead to potential information leakage between training and test sets due to the sequential nature of data. Therefore, a need for future work is clearly visible.

Even though the results are promising according to roc auc and precision scores, which are key for bug detection, due to potential leakage a verification of this approach is needed.

The main point in machine learning is the usability of the created models. To check the usefulness of the proposed solution, we performed surveys and had discussions with developers. The results clearly indicate that binary information about bug existence is insufficient.

6. Further work

Due to potential problems further work is needed.

The first issue to address is the verification of the current approach. We suggest finding ways to split data into training and test sets to reduce information leakage. For example, this can be done by considering the time of the commit, so as to split data into the portion introduced after and before a given date.

The next problem that should be tackled is how the labeling process is performed. This means that the algorithms from the SZZ class should be verified.

According to surveys there is also a need to improve the interpretability of the model's results. One possible way to do this is to manually create rules and suggestions from data, existing guidelines and good practices. Then we can associate them with a combination of features. Therefore the overall model will consist of two layers, one which is the model proposed in this article and a second layer combining features and predictions into a set of suggestions. The aim is that a developer sees only relevant suggestions.

References

[1] C. Jones, "A Short History of the Cost per Defect Metric," Namcook Analytics LLC, Narragansett, 2014.

[2] S. A. Slaughter, D. E. Harter and M. S. Krishnan, "Evaluating the Cost of Software Quality," *Commun. ACM,* vol. 41, pp. 67–73, 1998.

[3] C. a. G. B. a. S. E. Rosen, "Commit Guru: Analytics and Risk Prediction of Software Commits," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, Bergamo, 2015.

[4] M. a. H.-L. A. Nayrolles, "CLEVER: Combining Code Metrics with Clone Detection for Just-In-Time Fault Prevention and Resolution in Large Industrial Projects," in *ICSE*, Gothenburg, Sweden, 2018.

[5] Y. a. C. B. a. M. T. a. B. N. Jiang, "Comparing Design and Code Metrics for Software Quality Prediction," in *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*, Leipzig, 2008.

[6] E. a. H. A. E. a. A. B. a. J. Z. M. Shihab, "An Industrial Study on the Risk of Software Changes," in *ACM SIGSOFT 20th*

International Symposium on the Foundations of Software Engineering, Cary, 2012.
[7] "Klockwork documentation (Function- and method-level metrics)," Rogue Wave Software, Inc, [Online]. Available: https://support.roguewave.com/documentation/klocwork/en/10-1/functionandmethodlevelmetrics/. [Accessed 29 August 2018].
[8] T. Yin, "Lizard," open source, [Online]. Available: https://github.com/terryyin/lizard. [Accessed 29 August 2018].
[9] A. v. Cranenburgh, "Readability Package," Open source, [Online]. Available: https://github.com/andreasvc/readability/. [Accessed 29 August 2018].
[10] J. a. Z. T. a. Z. A. Śliwerski, "When Do Changes Induce Fixes?," *SIGSOFT Softw. Eng. Notes,* vol. 30, pp. 1–5, 2005.
[11] L. Breiman, "Random Forests," *Machine Learning,* vol. 45, pp. 5–32, 2001.
[12] F. a. V. G. a. G. A. a. M. V. a. T. B. a. G. O. a. B. M. a. P. P. a. W. R. a. D. V. a. V. J. a. P. A. a. C. D. a. B. M. a. P. M. a. D. Pedregosa, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825–2830, 2011.

About the author

In Nokia since 2014, mostly creating tools to improve overall quality. In the 2015 I have done my master's degree in Wroclaw University of Science and Technology. Since then I'm PhD student in the field of machine learning.

Jakub Kozerski
Engineer
MN BOAM Productization

Recommender systems: introduction with use-case idea

Weronika Białecka
Data Analyst
GS CS Network Engineering

NOKIA

1. Introduction

The idea behind recommender systems is rather simple – provide a user with new, not-yet-experienced items that may be relevant to the user's current task. A need for such systems emerged as more and more e-business services began to develop, offering the immense variety of products and services. Users started to find it difficult to search for items. They needed some kind of prompt that would help in making faster and better decisions. Recommender systems proved their role in dealing with information overload problem. Those systems generate recommendations using various types of knowledge and data about users, the available items, and previous transactions stored in customized databases. Out of those recommendations, a user can then choose the most suitable ones. This article will give you a brief introduction to typical machine learning algorithms used for building a recommender system. It does not contain the detailed description of the algorithms – it just introduces techniques commonly used within this area. Moreover, we will present the use-case idea for implementation of recommender system within Nokia.

1.1 Data and knowledge sources

In general, data used by recommendation system refers to three kinds of objects:

- items – products or services that are recommended. Recommendation system uses range of properties and features of an item depending on its complexity. Items with low complexity are for example books and movies, while insurance policies and jobs are more complex items.
- user – person that looks forward to a recommendation. Users can have very diverse characteristics and goals when using e-business services. Recommendation system can personalize result of recommendation (and to do so collects a range of information about user) or use non-personalized top-X selection.
- transactions-relations between users and items. The most popular transactions that recommender systems collect are ratings.

Data on those three objects can be used differently, depending on the type of the recommender system.

2. Recommendation techniques

There are different approaches to recommender systems that vary in terms of the addressed domain and the knowledge used. However, the main difference is the recommendation algorithm, i.e. how the system 'predicts' which items are worth recommending. In this article, we will present two most popular approaches for building such a system: content-based and collaborative filtering techniques.

2.1 Content-based approach

This technique is based on recommending items that are similar to those a given user has liked in the past, i.e. system analyzes a set of items viewed/rated by the user in the past and build a profile of user's interests based on the features of previously viewed items.

Figure 1 presents a high-level architecture of a content-based recommender. Content-based recommendation process consists of three main parts, each of which is performed by a separate component:

1. Content analyzer
It is a pre-processing part which is responsible for representing the content of items in a form suitable for the next processing steps. Techniques like feature extraction can be used within this component to represent data items in a suitable form and space (e.g. converting a text description to the vector of words).

2. Profile learner
This component collects data on user preferences and generalizes this data to construct user profile. This profile is usually created using machine learning algorithms that make possible creating a model of user interests based on items that one liked or disliked in the past.

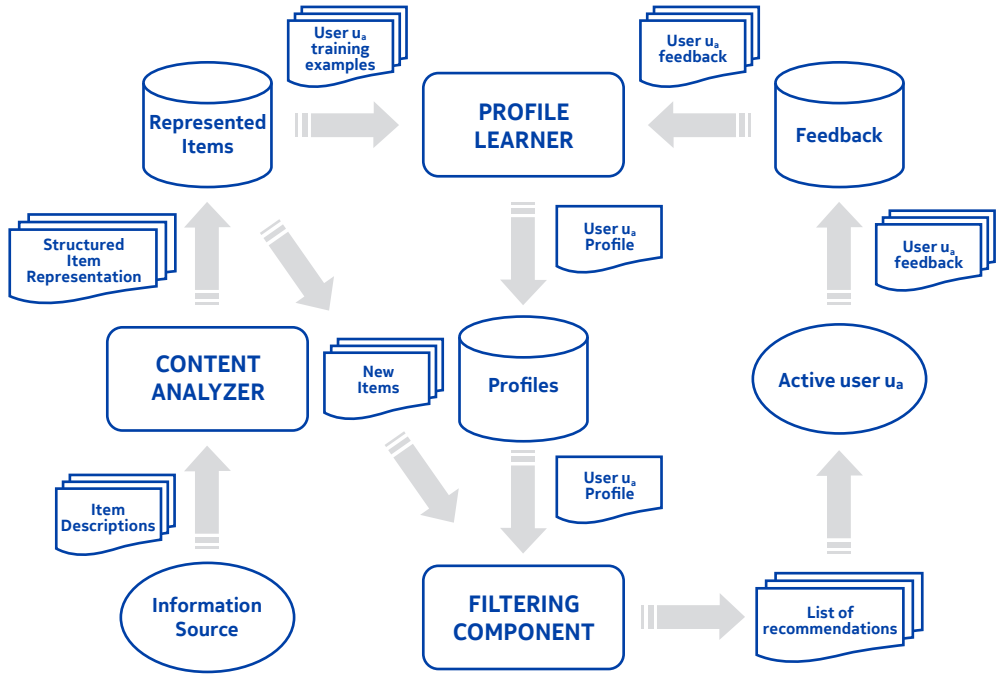
3. Filtering component
The last part of content-based system uses results of content analyzer and profile learner to suggest items worth recommending. It uses some similarity measures to find them.

2.2 Collaborative filtering approach

Collaborative filtering approach can be also called social filtering, as this approach is based not only on ratings of a given user *u*, but also on ratings of other users. The main concept behind this approach is that the user *u* might like/dislike the same things as user *v* given that their ratings from the past are similar. Collaborative filtering methods can be divided into two main groups:

1. neighborhood-based
Recommender systems from this group use user-item ratings to predict ratings for new items directly. This group can be divided into two sub-groups: user-based or item-based. When it comes to user-based system, it uses ratings of neighbors of user *u* to predict the interest for an item *i*. The neighbors of user *u* are those users

Figure 1 High-level architecture of a content-based recommender



whose rating patterns are the most similar to his rating pattern. On the other hand, item-based system predicts the rating of a user *u* for an item *i* based on the ratings of *u* for items similar to *i*. Two items are similar if several users rated the item in a similar way.

2. model-based
Model-based systems (called also latent factor models) learn the predictive model that estimates user-item interactions using several latent characteristics of items and users, e.g. category class of an item. Model is trained and used to predict the rating of user for the new item.

3. Machine learning methods applied to recommender systems

In the context of recommender systems, machine learning techniques can be used in every step of building such system: both preparation of data (for example dimensionality reduction) and analysis (recommending part).

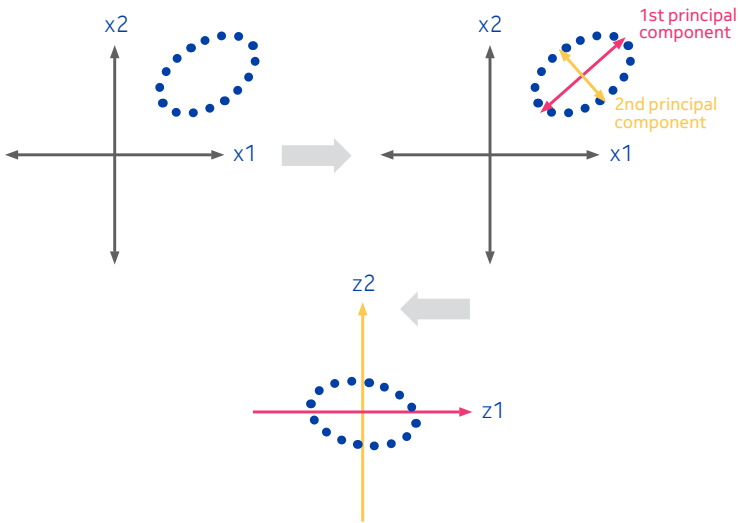
3.1 Dimensionality reduction

To build recommender system it is not enough to collect data – data should be also well prepared to be used. Hence, such actions as cleaning, transforming, filtering or handling missing data might be needed before actual finding recommendations.

We can define data as list of objects that have several attributes (properties). It is common in recommender systems to have a lot of attributes (each attribute is treated as a separate dimension; hence we often deal with a high dimensional space). Moreover, it also common to have very sparse information in that space, i.e. there are values for limited number of properties. It is hard to process data of that kind. Dimensionality reduction methods enable to overcome this problem by reducing number of features describing an object. The most popular machine learning method of reducing dimensions is Principal Component Analysis (PCA).

PCA is a method used to combine existing features of an object in order to transform the data into a new, lower-dimensional subspace. In other words, it just finds principal components of the dataset. By components we mean the directions, where there is the most variance (where the data is the most spread out). To find those components, we need to introduce eigenvectors and eigenvalues. An eigenvector is a direction, while an eigenvalue is a number describing how much variance there is in the data in that direction. The eigenvector with the highest eigenvalue is therefore the principal component.

Figure 2 Principal components in 2-dimensional space



Mathematically, we construct the covariance matrix of the original dataset that stores the pairwise covariances between different features. The covariance between two features x_j and x_k can be calculated as follows:

$$\sigma_{jk} = \frac{1}{n}\sum_{i=1}^{n}\left(x_j^{(i)} - u_j\right)\left(x_k^{(i)} - u_k\right),$$

where u_j and u_k are the sample means of features j and k. If we standardize the dataset, the sample means are zero. Covariance matrix can be then written as:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \sigma_{31} \\ \sigma_{21} & \sigma_2^2 & \sigma_{22} \\ \sigma_{31} & \sigma_{32} & \sigma_3^2 \end{bmatrix},$$

Number of eigenvalues and eigenvectors is equal to the dimension of covariance matrix (in this example it is three). Eigenvector v satisfies the following condition:

$$\Sigma v = \lambda v,$$

where λ is the eigenvalue (a scalar). Having the eigenpairs we choose k eigenvectors sorted by corresponding eigenvalues as principal components.

In order to reduce dimensionality, we can use ratings together with demographic features for users and demographic features for items. However, features need to be represented appropriately, i.e. represented as vectors.

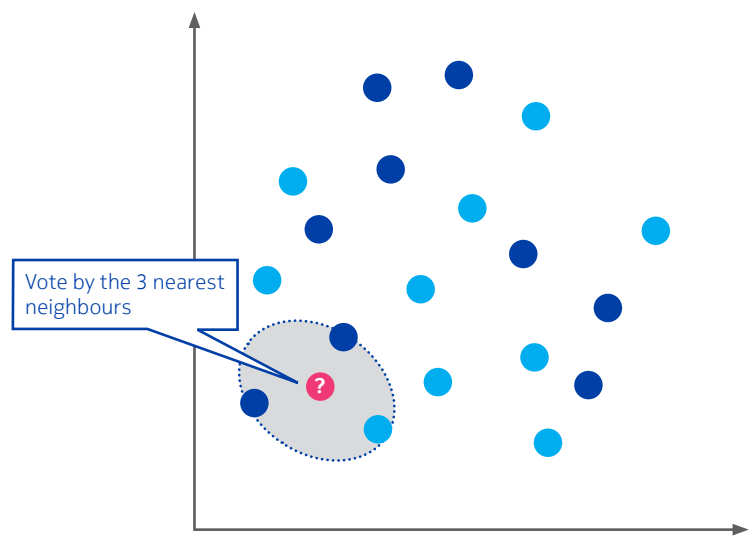
3.2 Classification

Classification is the process of predicting the class (label) of a given point, where the point has a set of features. Classification belongs to the category of supervised learning where the labels are known in advance and there exists a set of labeled examples which constitute a training set. We will describe several classification algorithms that are being used both in content-based and collaborative filtering recommender systems.

3.2.1 k nearest neighbors (kNN)

This classifier finds k closest points from the training records. Then, it assigns the class label according to the label of its k nearest neighbors which are chosen based on the calculated distance.

Figure 3 Example of k-nearest neighbors approach in two dimensions



There are several methods of calculating distance between points depending on the type of data, but the most popular one is the Euclidean distance:

$$d(x, y) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2}$$

We use Euclidean distance for real-valued samples, for example when the values are in centimeters. It is also important to standardize the data so that each feature contributes equally to the distance.

Another popular approach is to consider items as vectors of n-dimensional space and compute their similarity as the cosine of the angle that they form:

$$\cos(\mathrm{x}, \mathrm{y}) = \frac{(\mathrm{x} \cdot \mathrm{y})}{||\mathrm{x}||\,||\mathrm{y}||},$$

where denotes vector dot product and $||x||$ is the norm of vector x.

The third distance matrix commonly used in recommender systems is Pearson correlation, which calculates the linear relationship between objects:

$$d(x, y) = \frac{\sum(x, y)}{\sigma_x \times \sigma_y},$$

where $\sum(x,y)$ is the covariance of data points x and y and σ is the standard deviation.

Probably the most challenging issue in kNN is how to choose the value of k parameter. It is crucial to find a good balance between under- and overfitting.

Nearest neighbors method is one of the most popular approaches to collaborative filtering systems, as it is conceptually related to the idea of CF, i.e. finding like-minded users or similar items is essentially equivalent to finding neighbors for a given user or item. When it comes to items, this is also the approach that can be applied to content-based recommender systems.

3.2.2 Decision tree

This is a classifier in a tree structure. Each interior node corresponds to an attribute and each arc from a parent to a child node represents a possible value or a set of values of that attribute.

Decision trees are learned by recursively partitioning data into subgroups until those subgroups contain only instances of a single class. The choice of the term on which to operate the partition is generally made according to an information gain or entropy criterion. The entropy for all non-empty classes looks as follows:

$$I_H(t) = -\sum_{i=1}^{c} p(i|t)\ log_2 p(i|t),$$

where $p(i|t)$ is the proportion of the samples that belongs to class i for a particular node t. The entropy is then zero if all samples at a node belong to the same class and the entropy is maximal if we have a uniform class distribution. Therefore, we can say that the entropy criterion attempts to maximize the mutual information in the tree.

Information gain is defined as follows:

$$IG(D_p, f) = I(D_p) - \sum_{i=1}^{m} \frac{N_j}{N_p} I(D_j),$$

where D_p and D_j are the datasets of the parent and jth child node, $I(\cdot)$ is the impurity measure (entropy), N_p is the total number of samples at the parent node and N_j is the number of samples in the jth child node.

Decision trees can be used in content-based approach. A separate decision tree is built for each user and the features of items are used to build a model that explains user's preferences. The information gain of every feature is used as the splitting criterion.

When it comes to collaborative filtering, decision trees can be used in model-based approach. One possibility is to use a single customer as each instance in the training set. Then, the training set attributes refer to the feedback provided by the customer for each item in the system. In this case, a dedicated decision tree is built for each item. What is predicted here is the feedback provided for the targeted item.

1. other techniques
There are also other classification techniques that can be used for building a recommender system. For example, machine learning classifiers that were not described in this article are **Support Vector Machines** and **neural networks**.

3.3 Cluster analysis

Unfortunately, it can be hard to scale classification algorithms due to the amount of operations, e.g. computing pairwise distances (both for content-based and collaborative filtering approaches). A possible solution would be using PCA described earlier to reduce dimensionality of features. However, in collaborative filtering we can still have a problem with too many objects to compute the distance to. This is where clustering algorithms can improve efficiency, as the number of operations is reduced (we do not have to calculate pairwise distances between all objects). On the other hand, it can decrease accuracy. That is why the compromise between improved efficiency and a possible decrease in accuracy needs to be defined.

Clustering belongs to group of unsupervised ML algorithms and consists of assigning items to groups. The assumption is that objects within a group are similar to each other (by some distance metric). The goal is to minimize intra-cluster distances and maximize inter-cluster distances. The most popular algorithm from that group is k-means algorithm, but we can also use for example density-based clustering.

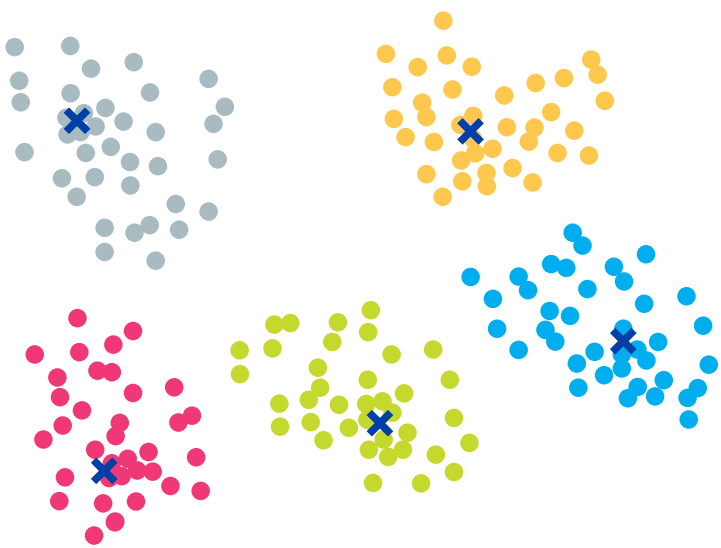
3.3.1 k-means

This clustering algorithm aims to partition objects into k clusters. The iterative algorithm has two steps:

- **Cluster assignment** – the algorithm goes through each of data points and depending on which cluster is closer, it assigns it to one of the clusters
- **Moving centroids** – algorithm calculates the average of all points in a cluster and moves the centroid to that location.

To calculate the distance, we use distance metrics (some of them were described in Section 3.2.1). The process is repeated until there is no change in the clusters or a maximum number of iterations is reached.

Figure 4 Example of k-means result



The most challenging part of this algorithms is that k parameter needs to be predefined. However, there is a method called elbow method that by calculating the sum of squared errors (SSE) for each k helps find this value of k parameter for which SSE is minimized. SSE is defined by following expression:

$$SSE = \sum_{i=1}^{n} \sum_{j=1}^{k} w^{(i,j)} \left|\left| x^{(i)} - \mu^{(j)} \right|\right|_2^2,$$

where $\mu^{(j)}$ is the representative point (centroid) of cluster j and $w^{(i,j)} = 1$ if the sample $x^{(i)}$ is in cluster j, otherwise $w^{(i,j)} = 0$.

Typical use of clustering in the context of recommender system is employing the k-means algorithm to help in neighborhood formation (both for users and items). Moreover, it can be also used as smoothing technique in which missing values for users in a cluster are replaced by cluster representatives.

4. Application in Nokia – webNEI tool

Recommender systems can be applied within Nokia. This section presents the use-case idea for building a recommender system for one of our internal platforms, i.e. webNEI.

Let us start with a short introduction what exactly webNEI is. This is a platform that was created to ensure quick access to all technical presentations (Network Engineering Information materials – NEIs). Those presentations refer to radio features that Nokia is offering to its clients. The platform supports community-based knowledge building. User can view the NEI online, download it and add to favorites as well as provide links to related materials.

A user that is searching for the appropriate presentation can filter results by several attributes of NEIs:

- domain (e.g. 5G, LTE, SRAN),
- release (depending on the domain),
- document type (e.g. single presentation, learning path),
- NEI scope (e.g. early birds, tooling, briefings),
- tags,
- top NEIs,
- searched text provided by a user.

When it comes to ratings, users have the possibility to rate the presentation using one to five stars.

Having stored the information about NEI itself, i.e. values of filtering attributes, as well as tags (e.g. IoT, Carrier Aggregation, Operability, Power Consumption, Security, Small Cells) and, what is even more important, users' behavior and ratings, we can build a recommender system. This system could recommend NEIs that are probably interesting for a given user.

In fact, webNEI is similar to YouTube. The difference lies in the definition of item being recommended. However, the mechanism is very similar. A user is searching for an interesting item to watch/read. Then, depending on how much user likes it, he spends more or less time watching the film (YouTube) or reads more or less presentation pages and even downloads it (webNEI). Moreover, similarly to YouTube, user can add an item to favorites and rate it.

YouTube specialists use deep neural networks to build their recommender system. However, this video-sharing website has much more users than Nokia's internal platform. Moreover, number of items is also much bigger. Hence, one could try simpler approach to build such a system for webNEI. It may turn out that one of the techniques presented in this article will be suitable enough to give satisfying items recommendations.

References

[1] Covington P., Adams J., Sargin E., Deep Neural Networks for YouTube Recommendations, 2016

[2] Raschka S., Python Machine Learing, 2015

[3] Ricci F., Rokach L., Shapira B., Kantor P. B., Recommender System Handbook, 2011

About the author

I have graduated my master studies in Applied Mathematics on Wroclaw University of Science and Technology in 2018, with specialization in Computational Mathematics. I have been working for two years in Network Engineering, where I analyze data on operators' network.

Weronika Białecka
Data Analyst
GS CS Network Engineering

Case Studies

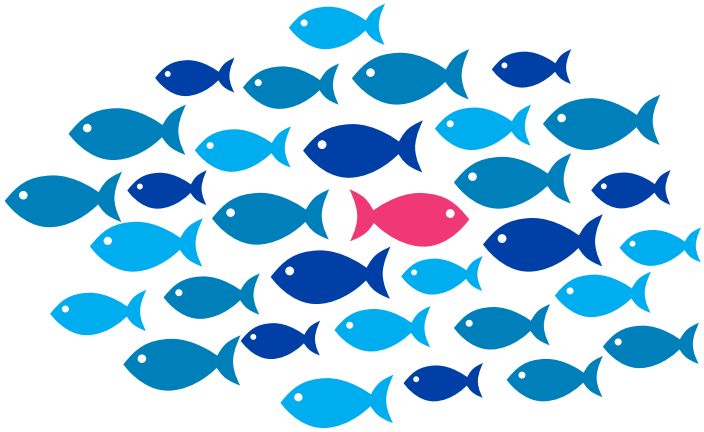
Anomaly Detection: application in telecommunication networks

Ewa Boryczka
Data Analyst
GS CS Network Engineering

NOKIA

Since telecommunication networks are becoming more and more complex, continuous network monitoring is essential in detecting and reporting failures as soon as possible. Unfortunately, the huge number of measurements that need to be monitored prevents manual investigations. Due to the specificity of network performance, two characteristics of its metrics can be indicated: seasonality and periodicity. Additionally, most valuable measures are the most variable as well (network throughput is a perfect example). In that case, setting a sensible threshold is very hard, and can be insufficient. This a chance for Machine Learning algorithms to supersede humans and do this job even better in an automated way.

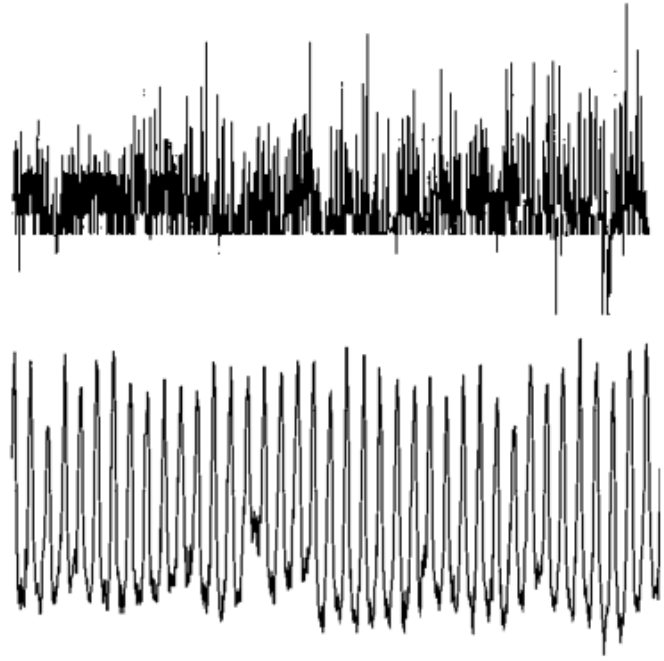
1. Define what to detect



Unfortunately, there is no clear mathematical definition of an **anomaly**. The question "What is an anomaly?" often yields a short answer "Something not normal". But this is not enough. There is no anomaly without a specified context. Look, is -1ºC an anomaly? In winter -1ºC seems normal, but the same temperature in the middle of summer holidays is, without any doubt, anomalous. It is crucial to define what you expect to detect, because you cannot detect something if it is not well-defined. The following definition of what constitutes an anomaly is pretty generic and popular in data mining, so let's use it for the purposes of this article:

> ***Anomalies are items or events that do not conform to an expected pattern or to other items present in a dataset.***

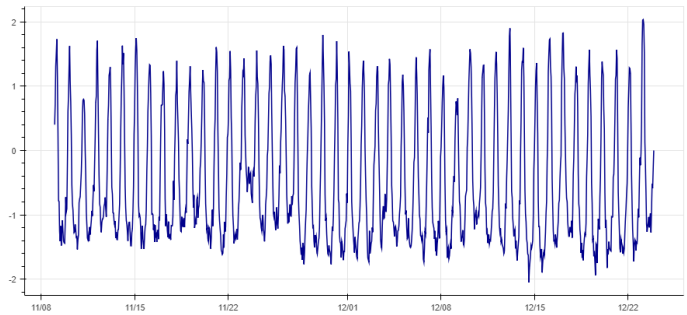
Now, warm-up! Look at the following images, and try to determine anomalies by eye.



It is not a trivial task even when a clear definition is given. The point is to find repetitive behaviors and use them as the expected pattern. Once we define the "normal", detection becomes straightforward. This is how our unsupervised algorithm for anomaly detection works.

2. Algorithm: step-by-step

The second part of this article is devoted to a step-by-step description of the algorithm. This section also provides examples based on the real-life KPI values, to engage a technical reader and to explain the "why" behind each step. The specific KPI is the average received signal strength indicator value for a physical UL control channel measured in the eNB. Our sample contains values of the KPI collected within an 1-hour time step for 45 days. We can expect periodicity of its values and repetitive character of the behavior over time. It can be inferred from the characteristics of real-life telecommunication networks. To confirm the hypothesis, it is necessary to view the KPI as a function of time in a graph. (Note: the values are shifted to zero for simplification of further computation; the amplitude remains unaltered).



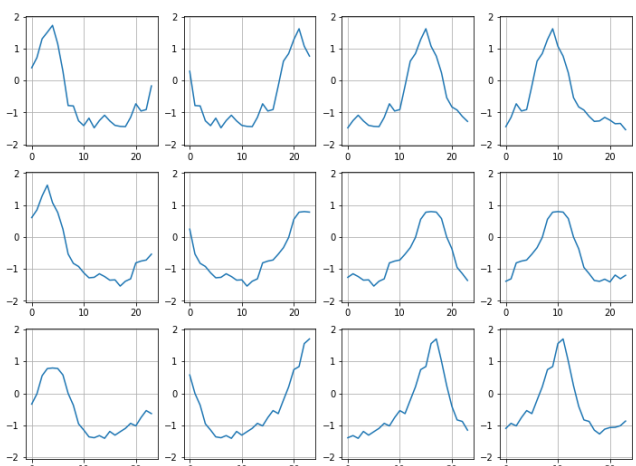
Quick visual investigation shows the regularities in the sample. It is therefore possible to describe a "normal" pattern of this KPI. Its values undulate with the 24-h period as it can be expected in the telco world.

2.1. Segmentation: sliding window

The proposed detector is based on historical patterns observed in the analyzed signal. It is established that the sample exhibits regular patterns with a 24-hour period. Now the task is to build a collection of all possible behaviors over time by cutting them in smaller chunks – in this case, chunks of length equal to 24 hours. It is important to collect **all 24-hour segments** to properly reconstruct the reference signal in further steps. For this purpose, a **sliding window** with the following parameters is used:

- length: $L = 24$
- sliding step: $S = 1$

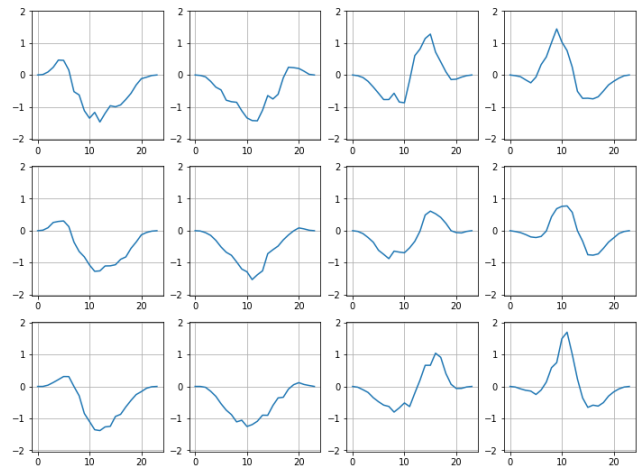
As the input for the segmentation step, the algorithm received the sample of length $N = 45 \times 24 = 1080$ and produced $(N - L)1/S + 1 = 1057$ segments of length $L = 24$. Let us take a look at some results:



All segments are undoubtedly wave-formed. It is important that they can be categorized into several groups based on shape. To prepare them for clustering and adapt to the reconstruction step, it is necessary to apply a **SIN-shaped filter**.

$$f(x) = \sin^2 x$$

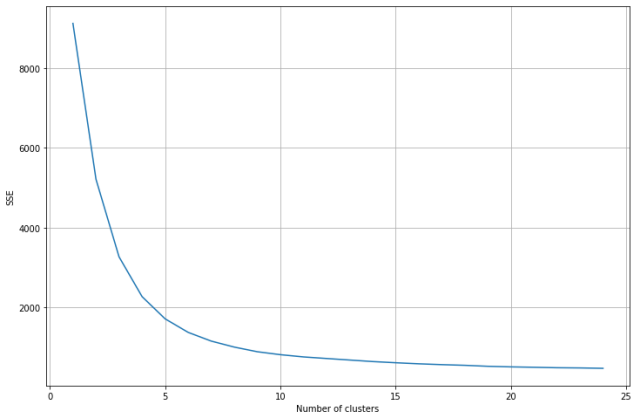
Filtering reduces the influence of segments' ends in the reconstruction. The middle part of each segment carries representative information about this chunk of the signal. The segments filter produces the following results:



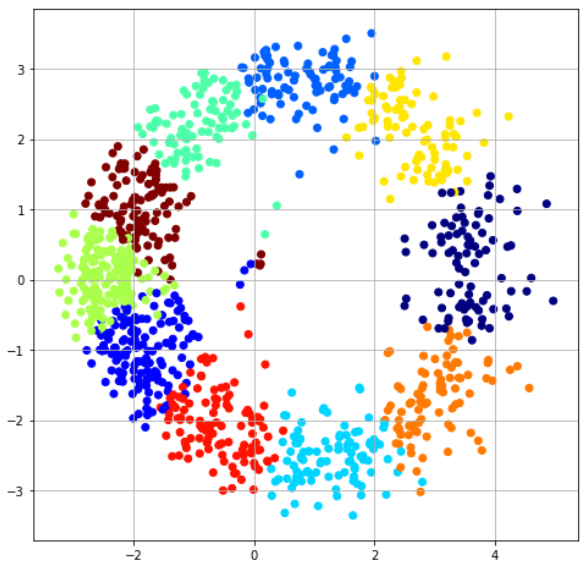
2.2. Clustering: K-means

The segmentation step produced chunks representing all possible behaviors observed within a 24-hour window. As the analyzed sample exhibits repetitive character, it is possible to group segments by similarity. One segment can be treated as a vector of length 24. Therefore, it is a typical clustering problem where one vector (segment) corresponds to one point in **a 24-dimensional space**. K-means algorithm is one of the most popular ML techniques. Such simple method gives us all necessary outcomes expected in our clustering phase. In this step centroids, the core part of **K-means**, will be used in the reconstruction of pattern signal. To determine the *K-number of clusters*, let's look at the **elbow curve**. The elbow curve is a graphical representation of SSE as function of the number of clusters K, where SSE is equal to the sum of distances of all samples to their cluster center.

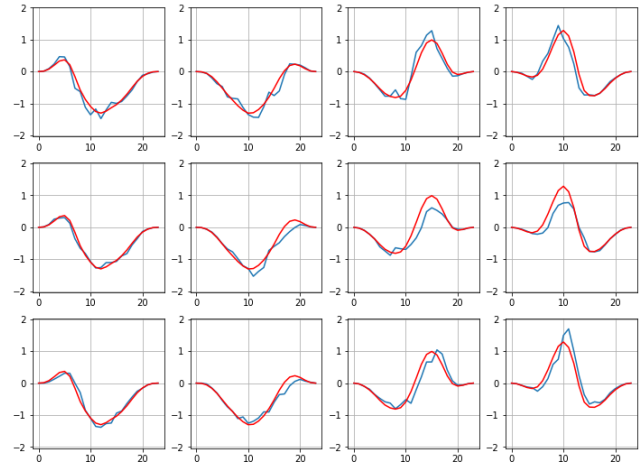
$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist(centroid_i, x)$$



The key goal is to find the lowest number of clusters with acceptable level of SSE. Let's take $K = 10$ and review the results of clustering. In order to visualize points and their assignment to clusters it is necessary to reduce dimensionality by Principal Component Analysis. Results in two dimensions are presented in the figure below. A quick visual investigation indicates a few outliers in our dataset – they represent the curious shapes of segments in our dataset.



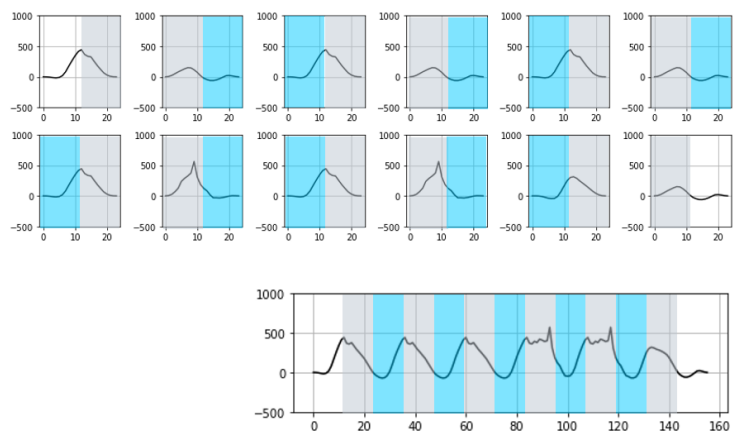
Each cluster has its centroid – it is the central point of the cluster and can be treated as the most representative point in this group. It will be important in the next step of our algorithm, where pattern signal is constructed on the basis of centroid shapes. Segments and centroids of their clusters are presented in the figure below.



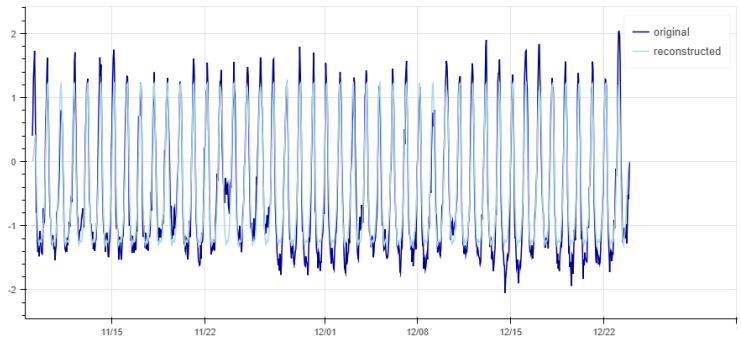
2.3. Reconstruction: pattern

Reconstruction is the main part of the proposed algorithm. The signal we reconstruct using cluster centroids represents the referral values for our sample and describes the "normal" behavior. In previous steps, we built collections of all possible behaviors over 24-hour period and grouped them into clusters with representative segments called centroids. Now we proceed to reconstruction, a process which consists of three steps:

1. Splitting the sample into overlapping segments by sliding window with length $L = 24$ and sliding step $S = L/2 = 12$. Apply SIN-filter (look: segmentation step)
2. Assigning each segment to the cluster and finding its centroid. The centroid represents the "normal" shape for this segment.
3. Joining centroids into one signal so that they overlap by a half of the length. The procedure is visualized in the figure below. Two grey parts of length 12 each produce a small piece of the reconstruction signal with the length of 12.



The final reconstructed signal for the analyzed sample looks like this:

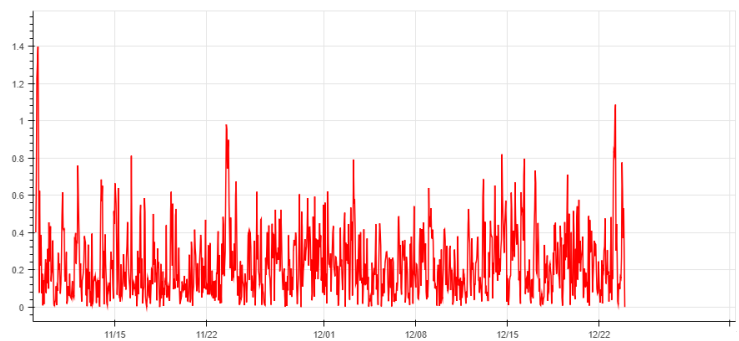


2.4. Reconstruction error analysis

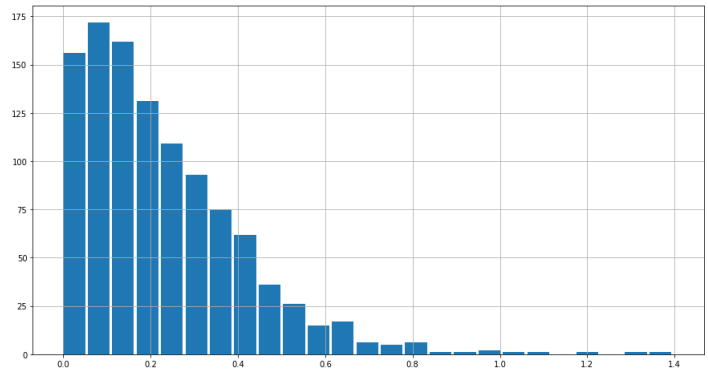
It is time for the final part of our algorithm: anomaly detection. The "normal" behavior of the analyzed KPI has already been defined in the Reconstruction step. Now our goal is to assign an **anomaly rate** to each real value of the KPI. The **anomaly rate** at a specified time point can be defined as a simple distance from the real value of the KPI to the expected value computed in the Reconstruction step – a longer distance means a more anomalous value observed at this time point.

$$r_{anomaly}(t) = |KPI(t) - Reconstruction(t)|$$

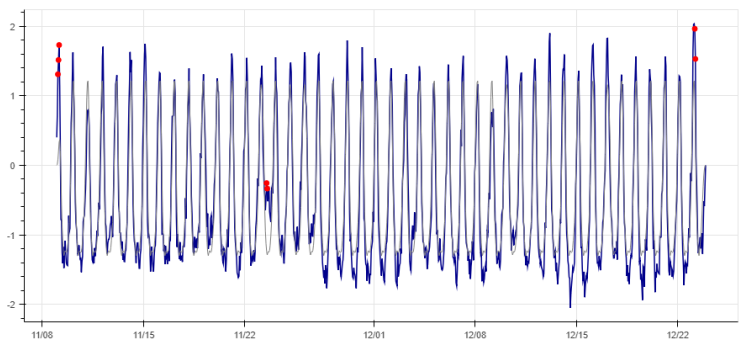
Let's confront the **anomaly rate** as a function of time with the reconstructed and observed values of the analysed KPI.



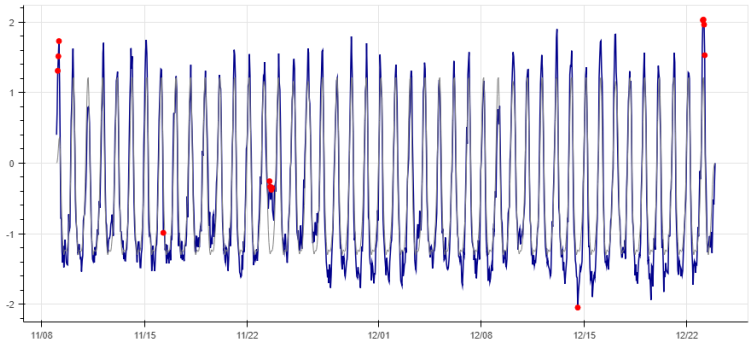
Deviations of observed values from predicted ones are mostly concentrated around zero. The majority of the values are located at the acceptable level – this does not indicate high deviation from the expected behavior. To detect anomalous values, it is necessary to determine an anomaly rate acceptance level for the analysed KPI. All points with anomaly rate higher (or lower) than the specified level should be reported as anomalous. To decide wisely, let's plot a histogram of anomaly rates and analyse the frequency of anomaly rate value occurrences.



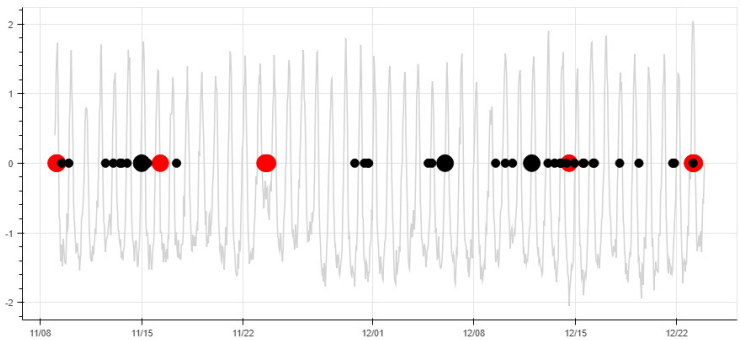
As we stated, most anomaly rates are at an acceptably low level. A few of them are higher than 1 – it is a significant deviation from the normal behavior, and should be definitely reported as an anomaly. A quick visual investigation of the histogram indicates that the anomaly rate slightly higher than 0.8 is very rare for the sample under investigation. Let's set an acceptance level equal to 0.9 and take a look at the anomalies detected with such parameter setting.



Anomalies have been detected in three areas. Two of them indicate an abnormal increase of KPI values beyond the expected range. The last anomaly we detected seems to be the most interesting one, as the it is the most difficult one to distinguish automatically. The value at this point is neither extremely high nor alarmingly low. This is the biggest advantage of the proposed algorithm: anomalies are detected with respect to the context in which they occur, with a sensitivity of the detector that can be tuned to match user needs. We can do it by re-selecting the anomaly rate acceptance level or by choosing how many of the most anomalous samples should be reported as anomalies. Let's choose 10% of the most extreme anomaly rate values. Then the acceptance level is equal to 90% percentile computed for our observations, ~0.81. Additional anomalies are now detected, e.g. the significant decrease of signal strength on 14th December.



3. Anomaly = Problem?



It is a common mistake to take anomalous behavior for a problem. In the figure above, you can see black dots drawn at all the time points in which the alarms for the selected eNB have been raised. Red dots indicate anomalous behavior detected by the proposed algorithm. There were some alarms raised around the time points with anomalies. There is also a time point marked as anomalous without an alarm – it is unexpected behavior but the KPI value does not indicate any alarming event on eNB. An average received signal strength was surprisingly high. It is an exceptional event over the analysed period and should be inspected further.

4. Summary

The project in which the proposed anomaly detection algorithm was applied concerned monitoring of over 300 key performance indicators at 15 eNB collected with an 1-hour time step. The monitoring produces around 120 000 samples for investigation each day and helps avoid manual exploration of the dataset. Over one day period, the detector indicates on average 100 anomalous values, and narrows down the area of manual investigation significantly. The proposed algorithm is a lightweight solution and does not involve any complex computation. Additionally, it can be extecuted for each sample separately, which means that it allows parallel computation for even faster results and makes continuous network performance monitoring possible.

References

[1] https://www.allerin.com/blog/machine-learning-for-anomaly-detection
[2] https://anomaly.io/anomaly-detection-clustering/
[3] Raschka S., Python Machine Learing, 2015

About the author

I have graduated my master studies in Applied Mathematics in 2018 at Wroclaw University of Science and Technology. This is where I started my journey into data science. At Nokia I work as Data Analyst in Network Engineering, where I utilize my analytical skills and statistical knowledge to generate insights from our customers data.

Ewa Boryczka
Data Analyst
GS CS Network Engineering

Software Development Aspects

NOKIA

Demystifying Machine Learning on Scale

Tomasz Przerwa
Data Engineer
GS CS Network Engineering

Mateusz Sołtysik
Specialist, Software Development
GS CS Network Engineering

NOKIA

The continuously growing popularity of the Internet of Things makes the amount of produced data increase exponentially over time. Based on IDC's research, "the world will be creating 163 zettabytes of data a year by 2025" [1]. The ability to process and analyze such volumes opens new, unknown so far, business opportunities. Usually, the volume of such data will not fit on a single computer node; moreover, it requires big-data-oriented solutions. In this article, we're going to discuss the building blocks[1], of highly-scalable, data-intensive systems and present a step-by-step practical analysis of such case.

1. The Building Blocks

As a general term, scalability may be understood in many ways. In this article, we define it as an ability to handle increasing amount of work through a growth of hardware resources. In practice, it is usually done by multiplying the number of computers involved in data processing (horizontal scaling).

In such distributed systems we face a challenge of forcing many computers to act like a single machine, i.e. to add up their resources and simultaneously work on the same problem.

The challenge spans many domains. We address each of them through separate Building Blocks, covered in next sections, which together amount to a complete recipe for machine learning in scale.

1.1 File Format

The Big Data revolution brings new file formats, designed to be more efficient in highly distributed ecosystems. We choose the file format basing mostly on its read/write performance, depending on the use cases.

Apache Parquet [2] is a column-oriented binary file format. Thanks to the support of indexing, which makes it effective for intensive reading, it is the most suitable format for data science.

Apache Avro [3] is a row-based serialization system. The design is better for intensive writing. Avro provides a more complex schema evolution. This is especially important for data whose schema changes over time.

1.2 Storage

A well chosen storage system is the foundation of data science systems. Regardless of the use case, we expect a highly available solution with an ability to grow.

One of the most widely used storage systems is Hadoop Distributed File System (HDFS) [4], released in 2006 under the Apache Foundation umbrella as a Hadoop library module. Its main task is to keep large files across multiple machines. If more space is needed, one can simply scale out the cluster horizontally. Due to its high-tunability, this system is well suited for complex use-cases.

Another alternative is Amazon S3 [5] (Simple Storage Service), object storage, released in 2006. The service provides easy access via a web interface. It can grow without limits, so it is applicable for storing large sets of unstructured data.

The choice between those two depends on resources used for your project. In cloud based solutions, like AWS, S3 is a pure winner because of low costs and easy setup. Using your own hardware is also a good choice if you need a specific configuration and/or there is some Hadoop know-how in your team.

1.3 Cluster Computing Framework

The nature of computing on multiple nodes requires a set of tools and algorithms that will run smoothly on a big data cluster. The algorithms in such circumstances must be parallel oriented.

One of the game-changing papers [9] in the big data world has been released in 2004 by Google. It describes a programming model for processing huge amount of distributed data using just two functions: *map* and *reduce*. Hadoop MapReduce is one of the most popular implementations of the paper's ideas.

Apache Spark [10] emerges from the limitations of the MapReduce framework. While the latter saves results to HDFS, Spark keeps them in memory. This brings a significant performance boost for iterative-based algorithms, where the same data needs to be read multiple times. Spark also offers abstractions which make easy transformations over the distributed dataset. It comes with several extensions, e.g. the Spark GraphX component, which adds support for graph-parallel computation, or Spark MLib, with scalable implementations of popular machine learning algorithms.

1.4 Resource Manager

Unfortunately, computing frameworks are not prepared to exist alone. They need an orchestrator responsible for i.e. task scheduling, resource management and nodes' synchronization.

Depending on your storage, we suggest two solutions: Apache YARN [6] and Kubernetes [7], both supported by Spark. Choosing between the two is a matter of storage: If we have an ambition to build and manage the distributed storage ourselves or we are tight to HDFS

1 Through building blocks we mean specific technologies and frameworks, which the authors consider to be particularly useful.

ecosystem, the reasonable choice is the former. Opposite, when we are given third-party distributed storage (like Amazon S3), the latter will do the job as and widely used orchestrator.

1.5 Interactive Notebooks

Interactive notebooks are a response to the increasing demand for tools allowing data scientists to quickly play with data and visualize the result. These tools offer a programming playground of sorts for fast prototyping and collaboration.

The most popular are Jupyter notebooks [11]. They are well established, especially in the Python community thanks to their simplicity and native support of the language. Unfortunately, their architecture is suited mostly for local environments.

Apache Zeppelin [12], on the other hand, is a project which targets mainly working with big data. It has native support for Scala language (and many others e.g. R, Python), and features Spark out of the box. Helium, its package manager, offers lots of addons available directly through a web panel. As well as Jupyter, it supports sharing notebooks with others.

2. Practical Example: NYC Transport Dataset

In this section we are going to present a model ML analysis. We have chosen the *New York City Bus Data* [13] dataset, located at the Kaggle platform. It contains over 5 gigabytes of records coming from buses' GPS sensors, recorded for four months in 2017.

The data will be converted to parquet file format and placed on HDFS. For this purpose, we have deployed a computing cluster on top of Apache YARN, using Apache Spark as the computing framework. As an interactive frontend, we are going to use Apache Zeppelin. For keeping clarity, we split the process into three different phases: Data Engineering, Data Exploration and, finally, Machine Learning.

2.1 Data Engineering

We start our journey from raw data located at the Kaggle website, so the first step is to download it to our machine. We do that by using Kaggle API [14] and running the following command:

```
$ kaggle datasets download -d stoney71/new-york-city-
transport-statistics
```

The data is compressed and, in such form, cannot be directly read by Spark, so we need to decompress it. Then we put it onto HDFS.

```
$ hdfs dfs -mkdir nyc-transport
$ hdfs dfs –mkdir nyc-transport/csv/
$ unzip new-york-city-transport-statistics.zip && hdfs dfs
-put mta_*.csv nyc-transport-data/csv/
$ hdfs dfs -ls -h nyc-transport-data/csv/
Found 4 items
-rw-r--r--   1 hdfs supergroup 1413655915 2018-12-24 09:37
nyc-transport-data/csv/mta_1706.csv
-rw-r--r--   1 hdfs supergroup 1356475974 2018-12-24 09:37
nyc-transport-data/csv/mta_1708.csv
-rw-r--r--   1 hdfs supergroup 1439214975 2018-12-24 09:37
nyc-transport-data/csv/mta_1710.csv
-rw-r--r--   1 hdfs supergroup 1332233477 2018-12-24 09:37
nyc-transport-data/csv/mta_1712.csv
```

So far so good, the data is successfully stored on HDFS in a CSV format. But querying CSV files is not efficient [15] when they are large. Spark is the most convenient way to do that, as it comes with a set of file manipulation methods for most of the data-related formats.

```
val sqlc = new org.apache.spark.sql.SQLContext(sc)
val csvPath = "nyc-transport/csv/mta_*"
val parquetPath = "nyc-transport/nyc-buses.parquet"
sqlc.read
    .option("header", "true")
    .option("inferSchema", "true")
    .csv(csvPath).write.parquet(parquetPath)
```

Now we are ready to explore!

2.2 Data Exploration

First, we load our dataset into Spark's **DataFrame**.

```
val nycTransportDF = spark.read
    .option("header", "true")
    .parquet("nyc-transport/nyc-buses.parquet")
```

We start from listing available columns and their data types. We can obtain the schema by using the **printSchema** method of the **DataFrame**.

```
nycTransportDF.printSchema
root
|-- RecordedAtTime: timestamp (nullable = true)
|-- DirectionRef: string (nullable = true)
|-- PublishedLineName: string (nullable = true)
|-- OriginName: string (nullable = true)
|-- OriginLat: string (nullable = true)
|-- OriginLong: string (nullable = true)
|-- DestinationName: string (nullable = true)
|-- DestinationLat: string (nullable = true)
|-- DestinationLong: string (nullable = true)
|-- VehicleRef: string (nullable = true)
|-- VehicleLocation.Latitude: double (nullable = true)
|-- VehicleLocation.Longitude: double (nullable = true)
|-- NextStopPointName: string (nullable = true)
|-- ArrivalProximityText: string (nullable = true)
|-- DistanceFromStop: string (nullable = true)
|-- ExpectedArrivalTime: string (nullable = true)
|-- ScheduledArrivalTime: string (nullable = true)
```

The first column is the timestamp of the measurement. The next eight columns give us general information about the bus line, such as line direction and number, first and last stop, and their coordinates. Next three contain the vehicle identifier and current GPS coordinates (at measurement time). The last five store information about the next stop such as name and distance.

Once we know the schema, we actually want to see the data. For that we can simply use Spark SQL combined with Zeppelin's supreme visualization abilities. We register the data frame to be able to query it in Spark SQL:

```
nycTransportDF.createOrReplaceTempView("nycTransportDF")
```

Using Spark SQL, we can also plot how many measurements we have each day. On the graph (**Figure 2**) we can observe strong weekly seasonality, although our dataset is not continuing – there is no data for July, September, and November.

Figure 1 Example output of Spark SQL query inside Zeppelin notebook

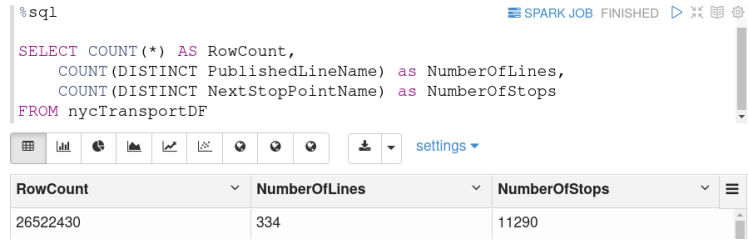


Figure 2 Visualisation of data sesonability

Spark's **DataFrame** provides a very useful **summary** method which gives us basic descriptive statistics for each column like *min*, *max*, *count*, *mean*, *standard deviation* and *quartiles*. Note that even though it works with all **Comparable** types, it was designed only for numerical types – for others, like string, it will show only minimum and maximum. Using the method, we conclude that there are some missing values in our data set. To investigate the scale of the problem, we have prepared a simple null-checker.

```
val count = nycTransportDF.count().toDouble
for (col <- nycTransportDF.columns) {
    val numOfNulls = nycTransportDF
        .select($"`$col`")
        .filter($"`$col`" isin
            ("NA", "NULL", "N/A", null, Double.NaN))
        .count
    println((col, numOfNulls, numOfNulls/count))
}
```

We found that almost 15% of **ExpectedArrivalTime** is missing, while for the rest of columns we have over 97% of non-null values.

2.3 Machine Learning

Once the data has been explored, we are ready to get some ideas for analysis. Our idea is to derive an empirical timetable for a given bus stop basing on the historical buses' arrivals.

To accomplish that task, first we need to filter rows which contain measurements when the bus is exactly at the stop.

```
val filteredDF = sqlc.sql("""
    SELECT
        PublishedLineName as line,
        RecordedAtTime as time
    FROM nycTransportDF
    WHERE NextStopPointName = 'HARWAY AV/BAY 37 ST'
    AND ArrivalProximityText = 'at stop'
""");
```

Before we create an actual model, we need to vectorize the time column. The first step in the vectorization process is feature extraction [16]. Some work has to be done to translate the time into format understood by Spark. At first we map days of week into binary vector of length 3. Then, we have to convert the hour into numbers. Our solution is to take seconds from midnight and wrap resulting number on unit circle. This will keep time continouity i.e. two hours that are close to each other in terms of time remain close after transformation:

```
import scala.math.{cos, sin, Pi}
import org.apache.spark.sql.functions.udf
def toSecs(d: java.sql.Timestamp) => (d.getSeconds
    + d.getMinutes * 60.0
    + d.getHours * 3600.0) / (24.0 * 3600.0)
val row2array = udf{ d: java.sql.Timestamp => {
    val dayVec = ("00" + d.getDay.toBinaryString)
      .takeRight(3)
      .map(_.toString.toInt.toDouble)
    val secsScaled = 2.0 * Pi * toSecs(d)
    dayVec union Vector(cos(secsScaled), sin(secsScaled))
}}
val vectorDF = filteredDF
    .withColumn("vector", row2array.apply($"time"))
```

Now, our goal is to rescale the dataset into a [0, 1] interval as well as split it into two: the training sample and the test sample. We also assign a numerical category for each bus line, to satisfy **LabeledPoint** types' requirements.

```
import org.apache.spark.ml.linalg._
import org.apache.spark.ml.feature.LabeledPoint
import org.apache.spark.ml.feature.MinMaxScaler

val lineMapping = vectorDF
    .select($"line".as[String])
    .distinct.collect.zipWithIndex.toList.toMap

val labeledDF = vectorDF.select(
        $"line".as[String],
        $"vector".as[Array[Double]])
    .map{ case (line, vector) =>
           LabeledPoint(lineMapping(line),
                        Vectors.dense(vector))
    }

val scaler = new MinMaxScaler()
    .setInputCol("features")
    .setOutputCol("scaledFeatures")
    .setMin(0).setMax(1)

val Array(trainingDF, testDF) = scaler.fit(labeledDF)
    .transform(labeledDF)
    .select($"label", $"scaledFeatures" alias "features")
    .randomSplit(Array(0.85, 0.15))
```

Finally, we can train and evaluate our model. We choose multi-layer feed-forward neural network as a classifier. For tuning hyper-parameters (in this case: size of layers) we use the combination of **ParamGridBuilder** and **TrainValidationSplit**.

```
import org.apache.spark.{ml => ml}
import ml.tuning.{ParamGridBuilder, TrainValidationSplit}
import ml.evaluation.MulticlassClassificationEvaluator
import ml.classification.{ MultilayerPerceptronClassifica-
tionModel,
MultilayerPerceptronClassifier}

val mlp = new MultilayerPerceptronClassifier()
    .setMaxIter(2000)

val paramGrid = new ParamGridBuilder()
```

```
    .addGrid(mlp.layers, Array(
        Array[Int](5, 25, 15, 10, lineMapping.size),
        Array[Int](5, 24, 20, 20, lineMapping.size),
        Array[Int](5, 40, 25, 20, lineMapping.size)
    )).build()

 val trainValidationSplit = new TrainValidationSplit()
     .setEstimator(mlp)
     .setEvaluator(new MulticlassClassificationEvaluator)
     .setEstimatorParamMaps(paramGrid)
     .setTrainRatio(0.8)

val model = trainValidationSplit.fit(trainingDF)

val Array(trainPrediction, testPrediction) =
    Array(trainingDF, testDF).map{df =>
       model.transform(df).select("prediction", "label")
    }

val evaluator = new MulticlassClassificationEvaluator()
    .setMetricName("accuracy")

println(
evaluator.evaluate(trainPrediction), evaluator.evaluate(test-
Prediction))

(0.8646288209606987, 0.8518184550646228)
```

We achieved accuracy of over 85%, almost as good as on the training set. A quite nice result for such a basic classification model.

3. Summary

We described a modern toolset used for handling big data analyses and demonstrated a model ML process from data input, through exploration, transformation, ending up with a ML pipeline.

This simple example shows how Spark MLib together with distributed storage could be used to solve various data science problems

References

[1] https://www.forbes.com/sites/andrewcave/2017/04/13/what-will-we-do-when-the-worlds-data-hits-163-zettabytes-in-2025/
[2] https://parquet.apache.org/
[3] https://avro.apache.org
[4] https://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist hadoop-hdfs/HdfsDesign.html
[5] https://aws.amazon.com/s3/
[6] https://hadoop.apache.org/docs/current/hadoop-yarn/ hadoop-yarn-site/YARN.html
[7] https://kubernetes.io
[8] https://www.infoworld.com/article/3118345/cloud-comput-ing/why-kubernetes-is-winning-the-container-war.html
[9] https://static.googleusercontent.com/media/research. https://google.com/en//archive/mapreduce-osdi04.pdf
[10] https://spark.apache.org
[11] https://jupyter.org/
[12] https://zeppelin.apache.org
[13] https://www.kaggle.com/stoney71/new-york-city-transport-statistics
[14] https://github.com/Kaggle/kaggle-api
[15] https://blog.cloudera.com/blog/2016/04/benchmarking-apache-parquet-the-allstate-experience/
[16] https://spark.apache.org/docs/2.4.0/ml-features.html

About the authors

Data Engineer, an expert in solving problems that stretch across software engineering, mathematics and data science.

Tomasz Przerwa
Data Engineer
GS CS Network Engineering

Full-stack Software Engineer, experienced in Big Data processing and analysis under DevOps practices.

Mateusz Sołtysik
Specialist, Software Development
GS CS Network Engineering

Software Development Aspects

Introduction to containerization for ML

Michał Pawlik
Site Reliability Engineer
GS CS Network Engineering

NOKIA

Abstract

The ML-powered applications of today have to run on a large scale. Let's take a closer look on what we can do to prepare our code for this challenge. This article will take you to the world of software containerization, describe how it helps to scale applications, and introduce a Nvidia-Docker variant of containerization that lets your containers benefit from the power of Graphics Processing Unit (GPU).

1. What is containerization

Machine learning (ML) libraries grow in complexity and so do their requirements. It often happens that dependencies change and we run into problems when it comes to development of multiple projects at the same time. This problem applies to execution environments, where one production server runs projects that often feature different library requirements, language interpreter version, or even multiple programming languages support.

This set of problems is not exclusive to the ML domain and has been faced by programmers and operators long before ML popularization. One approach was to distribute software as packages for package manager, that were then installed by the operators on production servers. This solution worked until two projects required specific, conflicting dependency versions.

One of possible solutions is virtualization. With virtual machines (VMs), we can set up a new application on separate VMs, giving them separate operating systems, all the necessary libraries and software, and then install our product. **Figure** 1 presents the stack built with VMs.

Figure 1 Stack with VMs

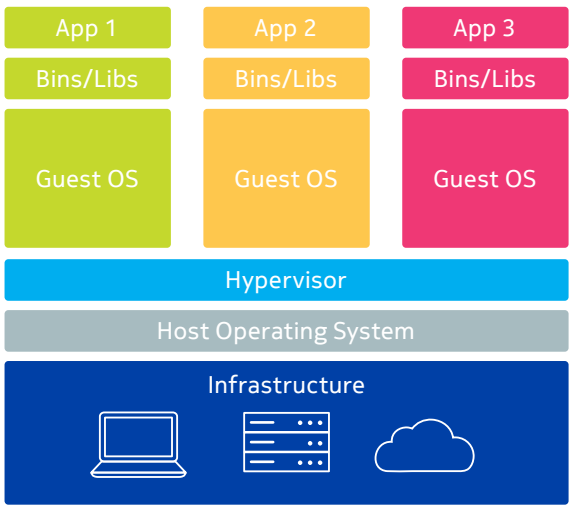
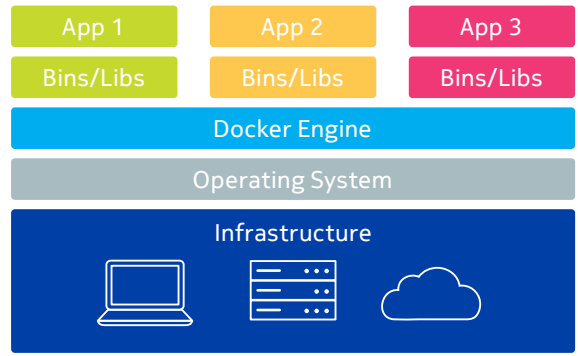


Figure 2 Stack with Docker containers



Although the use of VMs solves the issue, it creates other problems at the same time. Imagine working on three or four projects at a time. This would require developers to have a VM for every project. This causes huge overhead and makes it simple to desynchronize development requirements between environments. We need something lightweight and simpler.

Here, containers come to the rescue. How do they work? Simply put, they can be compared to very lightweight VMs with their own operating system and the application we have developed and want to run. They can be compared, but they are not VMs. In this article, we will focus on Docker containers. This is by far one of the most popular containerization engines and Nvidia provides a fork that enables lots of possibilities in terms of ML. Take a look at **Figure** 2 to see how execution stack changes with Docker.

To understand the difference and see what makes it possible for operating system to run containers, we need to dive into details. Docker uses Linux kernel features, like namespaces and control groups, to isolate processes. Thanks to those features, from the operating system point of view, a container is just an another process. On the other hand, from the process point of view, it looks like it's running on a separate operating system, as it's unable to see processes from other namespaces. We can also limit resource usage using control groups, so that multiple containers do not fight for memory or CPU availability.

2. Working with containers

Before diving into the details of building Continuous Integration/Continuous Delivery (CI/CD) pipelines, let's take a look the problem solved by Docker containers. Every developer either said or heard the infamous "It works on my machine" statement. Thanks to Dockerfile, which is adescription for building an application image, we can test our application in the same environment that it

is going to run in production, eliminating, or highly limiting, occurance of such situations.

Now that we know something about containers and understand some problems they solve, let's see how they influence the continuous integration pipeline. A usual pipeline is shown in **Figure 3**.

Figure 3 Continuous integration pipeline

This pipeline is abstract and tells nothing specific about the technology involved. Let's imagine what it would look like with Docker involved. Let's assume that all steps, except development, happen on external servers set up using tools like GitlabCI, Jenkins, Travis, or similar.

1. **Develop** – during the development phase, programmers can benefit from setting up local dependencies with no interference with the local system. This also isolates multiple versions of dependencies in case of working on multiple projects with different needs at the same time.

2. **Build** – developers very often build an application during the development phase, but whenever a new feature or bugfix is introduced, we need to build the version that's going to replace the current production release. Whenever a build happens on an external server, the whole development environment must be present on it. With Docker containers, we can run the build process in a container that satisfies all prerequisites.

3. **Package** – this step is dedicated to creating a deliverable out of a built application source code. The required outcome is a package or binary that's later going to be deployed either automatically or manually by the operations team. With Docker, we have one unified format independent of the technology used during development. The format is a docker image, fundamental and only block required for running docker container.

4. **Test** – similarly to build step, tests can be run inside docker container (or multiple containers, depending on tests performed) to make them more portable, meaning executable by any currently available server, and scalable if necessary, meaning that long running tests can be distributed. If this step fails, application deployment is prohibited.

5. **Deploy** – with Docker, the deployment process is unified. A Docker image prepared in the package step is deployed to the target platform – be it either a server with a Docker daemon, Docker Swarm, Kubernetes, OpenShift, or a Rancher cluster.

6. **Operate** – operating with Docker makes operators focus on the platform instead of the details specific to the application technology, enables scalability, and improves maintainability.

This is how Docker influences the daily work of a programmer. With this knowledge, we can proceed to containerizing an exemplary application to get practical overview of the subject.

3. Containerizing an exemplary application

In this section, we are going to write and containerize a simple application that calculates the birthday problem probability[1]in. It "*concerns the probability that, in a set of n randomly chosen people, some pair of them will have the same birthday*".

Before starting, make sure you have Docker installed. Installation instructions vary across environments, but are well described in the documentation [2].

Containerizing an application means creating a Docker image of our program. To do that, we need two things: our program and a Dockerfile – a file describing the procedure of building our own image. Let's start with writing simple code for solving the birthday problem. The code is presented in **Listing 1**.

Listing 1 Birthday problem solution

```
#!/usr/bin/python3
from __future__ import division

from math import e
import sys
YEAR_DAYS = 365

def help_and_exit():
print(f"Calculate probability of pair in group of N having
 the same birthday.\n\nUsage:\n\tpython3 {sys.argv[0]}
 number_of_people")
exit()

def birthday_collision_chance(people: int, year_days: int) ->
float:
return 1 - e **(-people**2/(2*year_days))

def main():
```

```
if len(sys.argv) != 2:
help_and_exit()
try:
number_of_people = int(sys.argv[1])
except:
help_and_exit()
print(f"Probability is: {birthday_collision_chance(number_of_
 people, YEAR_DAYS)}")

if __name__ == '__main__':
main()
```

The core of the program is the **birthday_collision_chance** function that represents the probability calculation according to the following equation:

$$p(n,d) \approx 1 - e^{\frac{-n2}{2d}}$$

For a detailed description of the equation please refer to source article[1]. **python3 main.py 23**, which results in: **Probability is: 0.5155095380615168**.

Now that we have a working example, we can create a Dockerfile. Before writing, let's consider the requirements. Our application is very simple, it doesn't need any external libraries and the only dependency is python3 installed, the 3.7 version should be enough.

We usually start writing Dockerfiles by looking for existing and supported images on Docker Hub[3]. We can use the official Python docker image[4]. Images are tagged for the sake of versioning, we are looking for the smallest image possible that fits our needs.

Now we can proceed to writing the file itself. Dockerfiles are text documents containing commands for the Docker build tool. Details on syntax and all commands are available in documentation[5]. Ready Dockerfile is presented on **Listing 2**.

Listing 2 Dockerfile contents

```
FROM python:3.7-alpine

COPY ./main.py /opt/main.py
ENTRYPOINT ["python3", "/opt/main.py"]
```

Figure 4 Selecting a base docker image

3.6.6-alpine3.8	29 MB	4 days ago
alpine	30 MB	4 days ago
3-alpine	30 MB	4 days ago
3.7-alpine	30 MB	4 days ago

Three clauses are contained within our Dockerfile. First comes the **FROM** clause that tells Docker which base image to use. Think of Docker images as a set of layers, one on top of another. Our image will be a result of putting our application on top of the **python:3.7-alpine** image.

The second command is **COPY** – it stands for putting our code inside the image, specifically into the **/opt/** directory. Why **/opt/**? Docker containers are sharing kernel with the Linux host and that implies following the Linux filesystem hierarchy. According to the Filesystem Hierarchy Standard[6], it's the most convenient location for our program.

Using the last command, **ENTRYPOINT**, we can tell Docker what should happen whenever a Docker container based on our image is launched. It expects a list of arguments, where the first argument is usually the binary to be called and all the others are parameters for the binary. In our case, it's python3 binary followed by the full path of our program placed inside the image.

We have developed the description of our Docker image. Now, we are going to build the image and run a container from it. For this, we need command line and Docker installed in our system. To build our image, run **docker build -t birthday-problem** in the terminal, where *birthday-problem* can be replaced with anything as it's just a name for our image. Before running this command, make sure you have navigated to the directory containing our project files, main.py and Dockerfile.

As we have our image ready, let's run a container from it. For that, we are going to use the **docker** run command as follows: **docker run --rm birthday-problem 23**. As expected, the output is **Probability is: 0.5155095380615168**. The **--rm** flag informs Docker to remove the container after execution, birthday-problem is the name of the image to be used, and 23 is an input parameter expected by our program.

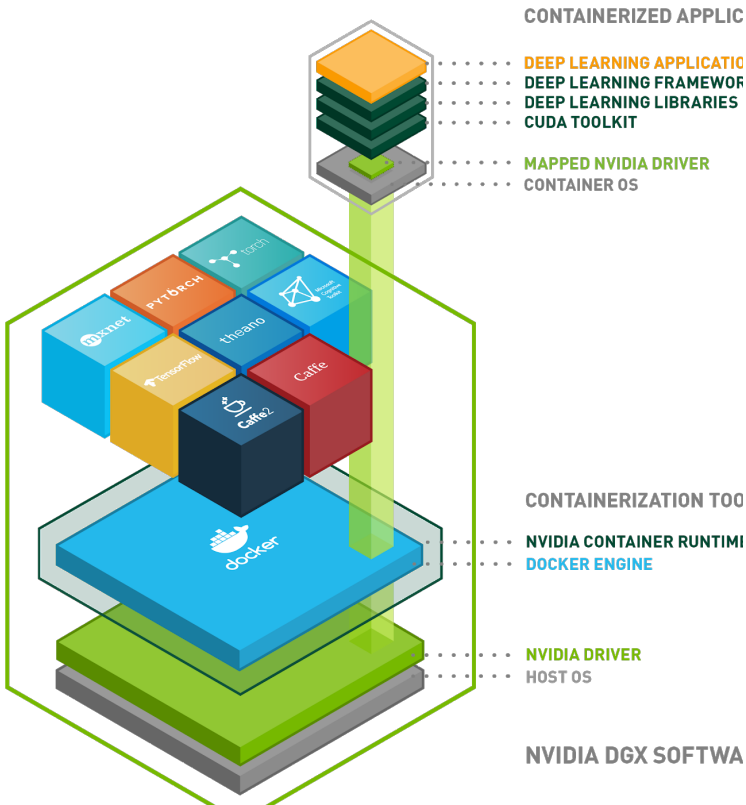
4. Using nvidia-docker image

Let's dive into how Docker can be applied to ML. To understand why we need Nvidia Docker, it's best to first identify the problem solved with this specific technology. In ML, we usually perform calculations over large vectors or matrices which require quick execution of many very similar operations in parallel. For this kind of computing, it's best to use hardware built of hundreds of thousands arithmetic processing units known better as a graphics cards. To experiment with ML, we need strong GPU.

Aware of the GPU role in ML, let's see how it affects containers. Every time we want to use external hardware from our computer, driver software is required. Due to high level of context isolation provided by Docker, which is one of its features, accessing the graphics card directly becomes difficult. To avoid struggle, we're going to learn how to solve the problem.

The solution is called nvidia-docker and is hosted on GitHub. It's also the place where we can find installation instruction[7].

Figure 5 Nvidia Docker architecture



Instead of writing our own application to see effects of GPU computation from a Docker container, we are going to use one of the samples provided by Nvidia in their official repository[8]. Nvidia also provides their own Docker registry where we can find more images that benefit from the power of GPU[9].

We are going to run the Hashcat utility benchmark on using GPU. Hashcat is a very fast password recovery tool that features in Central Processing Unit (CPU) and GPU variants. The tool is available on GitHub. The Nvidia samples repository already contains a Dockerfile for this project.

Let's build the image with the **`docker build -t hashcat`**. command. When the build is finished, we'll have a local *hashcat* image. First, let's try using it in the standard manner, without using Nvidia drivers. For that, we're issuing the **`docker run --init -ti --rm hashcat`** command. This does not work, and results in the following response:

Listing 3 Hashcat container launch results – no Nvidia docker

```
hashcat (v4.2.1-28-ga3ee4d7) starting in benchmark mode...

Benchmarking uses hand-optimized kernel code by default.
You can use it in your cracking session by setting the -O option.
Note: Using optimized kernel code limits the maximum supported
password length.
To disable the optimized kernel code in benchmark mode, use
the -w option.

clGetPlatformIDs(): CL_PLATFORM_NOT_FOUND_KHR
```

That means that the application has no access to graphic card drivers. We can fix it by providing the *runtime* flag. Let's now run it with the Nvidia Docker mode by executing: **`docker run --runtime=nvidia --init -ti --rm hashcat`**. It results in the following output:

Listing 4 Hashcat container launch results – with Nvidia docker

```
hashcat (v4.2.1-28-ga3ee4d7) starting in benchmark mode...

Benchmarking uses hand-optimized kernel code by default.
You can use it in your cracking session by setting
the -O option.
Note: Using optimized kernel code limits the maximum supported
password length.
To disable the optimized kernel code in benchmark mode, use
the -w option.
```

```
OpenCL Platform #1: NVIDIA Corporation
======================================
* Device #1: GeForce GTX 1080, 2029/8119 MB allocatable, 20MCU
* Device #2: GeForce GTX 1080, 2029/8119 MB allocatable, 20MCU

Benchmark relevant options:
===========================
* --benchmark-all
* --optimized-kernel-enable

Hashmode: 0 - MD5

Speed.Dev.#1.....: 26453.7 MH/s (49.91ms) @ Accel:128 Loops:512
Thr:1024 Vec:4
Speed.Dev.#2.....: 26513.6 MH/s (49.81ms) @ Accel:128 Loops:512
Thr:1024 Vec:4
Speed.Dev.#*.....: 52967.3 MH/s

...
Started: Tue Aug 28 08:09:02 2018
Stopped: Tue Aug 28 09:07:46 2018
```

In this case two GeForce GTX 1080 graphics cards were detected and used as you can see in **Listing** 4. The benchmark has been finished properly.

5. Summary

Containerization solves problems in both worlds: software development and operations. Docker containers can be very useful for ML purposes as they can:

- Simplify software configuration
- Separate environments
- Isolate software dependencies
- Make the deployment process very easy
- Enable resource management when operating an application
- Train multiple models on the same server in parallel without one project influencing another

Resources

[1] https://en.wikipedia.org/wiki/Birthday_problem
[2] https://docs.docker.com/install/
[3] https://hub.docker.com/
[4] https://hub.docker.com/_/python/
[5] https://docs.docker.com/engine/reference/builder/
[6] http://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.html
[7] https://github.com/NVIDIA/nvidia-docker/wiki/Installation-(version-2.0)
[8] https://gitlab.com/nvidia/samples
[9] https://ngc.nvidia.com/registry/

Worth reading

1. https://blog.openshift.com/use-gpus-openshift-kubernetes/
2. https://devopscube.com/what-is-docker/
3. https://devblogs.nvidia.com/maximizing-nvidia-dgx-kubernetes/
4. https://devblogs.nvidia.com/gpu-containers-runtime/
5. https://devblogs.nvidia.com/nvidia-docker-gpu-server-application-deployment-made-easy/
6. https://opensource.com/resources/what-are-linux-containers
7. https://success.docker.com/article/introduction-to-user-name-spaces-in-docker-engine
8. https://github.com/hashcat/hashcat

About the author

Site reliability engineer in Nokia GS CS Network Engineering. Software developer professionally for 6 years. Author of publication for Programista magazine. Advocate of Scala programming language and functional programming. Linux user. Bachelor of computer science at Wrocław University of Technology.

Michał Pawlik
Site Reliability Engineer
GS CS Network Engineering

Software Development Aspects

GPU computing power for Machine Learning

Wojciech Penar
DevOps Engineer
GS CS Network Engineering

NOKIA

Abstract

Display adapters have gone a very long way from simple computer display interfaces to modern graphics processing units. At some point, the computation power of specialized processors dedicated to graphics processing (so called graphics processing units, GPUs), outperformed the power of computers' main processors (central processing units, CPUs). The idea of using a GPU for scientific calculations was an obvious consequence. In fact, there are dedicated general-purpose graphic processing units (GPGPU) add-on cards, equipped with dedicated processors, similar to those used in top "graphic cards", but without output connectors. They found a wide range of applications, ranging from 3D visualizations to ML and AI solutions. Let's take a closer look at devices that drive modern high-performance computing.

1. Introduction

When talking about a GPU and computations, the first thing that comes into one's mind is cryptocurrency "mining". Press reports about shortages on the GPU market caused by the popularity of cryptocurrencies revealed to a wide audience that modern graphic cards may be used for tasks not related to graphics. In science-related and high-performance computing circles, such applications have been known for about decade. While GPUs are primarily dedicated to graphics processing, their processing units have many advantages over "typical" general purpose CPUs. Let's take a closer look on the development of modern GPU hardware and its use in applications other than graphics processing. While there are other GPGPU and parallel computing hardware vendors, this article is based on the example of Nvidia GPUs.

2. Brief history of computation and GPUs

The first specialized graphics processing hardware appeared in the 1970s to facilitate arcade game development. At its advent, nobody suspected that in just about three decades it would revolutionize computing. Until 1990, the main rationale for graphics processing offloading was to save on expensive memory. With the appearance of software that made extensive use of GUI, those simple *display controllers* were replaced by *graphic accelerators* – at first accelerating 2D graphics, then supporting video and 3D graphics. In the 1990s and 2000s, also *application accelerators* were used, but their use was limited to high performance workstations used in engineering – not something that could be found in a popular personal computer.

The most common application for 3D graphics in the PC world were games. Like nearly 30 years earlier, once again the arcade business pushed technology forth. Modern shooters or RPG games required nice-looking 3D graphics – nobody wanted to play in a pixelated world anymore (with some exceptions, such as Minecraft). In most cases, "nice" meant "realistic". The first step was achieved using texturized models – a 3D object was divided into a number of polygons, usually triangles. Then each polygon has assigned bitmap, transformed according to the angle of view, scene lighting and other conditions. The more and the smaller polygons are used, the more detailed shape could be rendered, but more computing power is needed to compute each polygon's appearance and visibility. To facilitate this operation, shader units were introduced.

The idea of shaders was introduced in 1988 by Pixar's RenderMan Interface Specification, Version 3.0. In technical terms, shading required large amounts of computation because each pixel in the final image was the result of a series of transformations. It allowed rendering of almost photographic quality scenes, but at the price of computation power and time – it was done by software means, later accelerated with the support of dedicated hardware. It is not surprising then that the use of this technique was limited to entertainment industry professionals.

The breakthrough came in the beginning of the 2000s, when the growing demand for realistic 3D games led to the introduction of programmable shader units into consumer-grade graphics cards. Rapid development led to the introduction of more and more powerful processing units, capable of performing multiple floating-point operations in parallel, or even to operate on vector data.

Yet powerful but dedicated to entertainment – from the scientific point of view, such computing power was wasted. There were some attempts to transform computational problems to the image-processing domain, process them as images and then transform back to their original domains. Limitations of the OpenGL library and complexity of required transformations limited applications of such approach, but the first steps were made.

In 2007, NVidia released a new product – NVidia Tesla GPGPU cards, accompanied by NVidia CUDA platform. While Tesla cards internally reassembled the top line of corresponding regular graphics cards (same processing unit family), they lacked display connectors – they were dedicated solely to provide computing power. CUDA, then distributed as single software bundle, contained tools and libraries required to write, compile, and run programs that could be executed with the support of GPU. In contrast to previous approach (exploiting the OpenGL library), writing programs intended to be run on GPU no longer required knowledge of graphics processing algorithms.

New multi-core processing units capable of floating-point operations together with an approachable set of software tools, such as NVidia CUDA and OpenCL, finally gave GPU computing resources to other applications than graphics processing. GPGPU was born.

While being a work-horse for many computation fields, GPGPU cards remained in the shadow. They speeded up computations in mechanical analysis, fluid mechanics, thermodynamics, biotechnology, and various modelling activities from visualization rendering for architects to geological analysis and real-time 3D modelling. Their use for non-graphics applications became more widely known after the cryptocurrency outbreak. High return on investment in Bitcoin "mining" caused a shortage of top consumer-grade graphic cards – they were ten-fold cheaper than dedicated hardware, but had similar computation power.

The GPGPU development has not been stopped yet. While introducing real-time raytracing capabilities, recent processing units are also fitted with *tensor cores*, capable of multiplying matrixes in single step rather than multiple "multiply and accumulate" steps. This feature is explicitly intended to facilitate machine learning applications – namely deep learning. In practice, it means computation speeds up.

Today, we can observe two main segments on the GPGPU market: customer-grade cards, such as Radeon series by AMD and NVidia GeForce series, intended for use in commodity hardware and cards intended for workstation/HPC use, such as AMD FirePro, AMD Radeon Pro, and NVidia Quadro as graphic cards accompanied by AMD Radeon Instinct and NVidia Tesla – both dedicated to computation. While top consumer-grade GPGPU hardware is still close to professional hardware, some vendors limited the use of hardware intended for gaming in computation environments and enforced it by including additional checks on driver loading.

3. Architecture of a modern GPU

As stated before, GPUs were designed to process large amounts of specific data in short time. The main mean to achieve this goal was parallelism – execution of graphics processing algorithms simultaneously on multiple cores. In opposition to usual concurrency, processing is performed in Single Instruction, Multiple Threads (SIMT) manner – Same Instruction, Multiple Data (SIMD) stretched over multithreading. In the SIMD model, the same operation is performed on multiple data flows at a time – as many as processing units. Introducing SIMT allows multiple threads to be executed on a single core, but in opposition to HT technology known from CPU, threads are executed in lock-step, meaning that the same instruction is performed in each thread at a time. Since there is no concurrency (so computations on different data flows are independent), the impact of the parallel slowdown phenomenon may be neglected. This also implies the most beneficial class of algorithms that may be run on GPGPU – these are so called "perfectly parallel" (or "embarrassingly parallel") workloads. Such workloads include both graphic processing (each pixel may be evaluated independently), password cracking (multiple hashes can be calculated at the same time) or cryptocurrency mining (which reassembles password cracking from the computation point of view). Many ML and AI algorithms may also be represented as a series of independent computing operations and could be processed using GPGPU. Especially neural networks and deep neural networks consist of many identical perceptrons organized into layers. Each such perceptron may be represented by a single computation thread and having ability to run 4096 or more such threads simultaneously reveals the true computing power of GPGPU.

Figure 1 GV100 (codenamed Volta) internal organization *(nVidia 2017)*

Looking under the hood (or better – heatsink) of a typical GPGPU, we find a processing unit accompanied by a large amount of memory. Computing cores are organized hierarchically. In case of NVidia Volta architecture (GV100 GPU), there are six groups, so called GPU Processing Clusters (GPC). Each GPC consists of seven Texture Processing Clusters (TPC). Each TPC consists of two Streaming Multiprocessors (with 84 SM in total). Each SM is built of 64 32-bit floating point cores, 64 32-bit integer cores, 32 64-bit floating point cores, 8 Tensor cores, and 4 texture units. It gives 5376 32-bit FP cores, the same amount of 32-bit integer cores, 2688 64-bit FP cores, 672 tensor cores, and 336 texture units. Internally, each SM is partitioned into 4 processing blocks. Each block has its own warp scheduler, capable of executing 32 threads per warp (and 2048 threads per SM in total) – such architecture is designed to facilitate parallel computations needed by graphics processing, but tensor cores are provided solely to fulfill ML needs.

To feed data into cores, eight memory controllers are employed, each equipped with a 512-bit memory bus, giving 4096-bit width of memory interface in total. Such interface offers memory bandwidth of 900GB/s compared to 70GB/s achievable by modern CPUs. Similarly to CPUs, GPGPUs contains their own cache hierarchy. Each SM contains an L1 cache, divided into data and instruction cache. There is also the L0 Instruction Cache, dedicated to each processing block inside an SM. The L2 cache is shared among the whole chip.

While the above data describe capabilities of the GV100 chip, actual computation resources available may vary due to different chip configuration – in case of Tesla V100, "only" 80 SM remains active, limiting the number of available cores to 5120 FP32 and INT32 cores, 2560 FP64 cores, and 640 tensor cores. Tesla V100 has 16GB of memory.

4. How fast is it?

One of most frequent questions on the use of GPGPUs in AI/ML applications is "Does it really work?". While graphics processing performance is clearly visible (each new GPU card is benchmarked against most requiring games for framerate), benchmarks against AI/ML workloads are not as common.

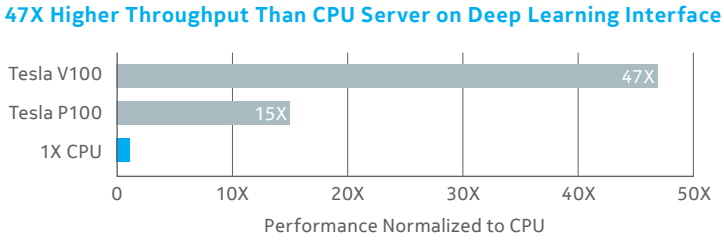
Scientific reports on GPU usage date as far as 2005 (P. Y. Simard 2005), reporting popular GPUs of that time being 3 times as fast as a 3GHz Pentium 4 CPU (two models were tested: ATI Radeon X800 and NVIDIA GeForce 6800 Ultra). More recent works (Lazorenko 2017) shows GPGPU accelerated solutions to be even 15 times faster than modern CPUs. How far could we go?

Looking into NVidia press releases, one can find the results, presented in **Figure 3**. Note that results are based on inference (classification with the use of trained neural network), which is cheaper than learning in terms of computational effort.

Figure 2 GV100 Streaming Multiprocessor internal organization *(nVidia 2017)*

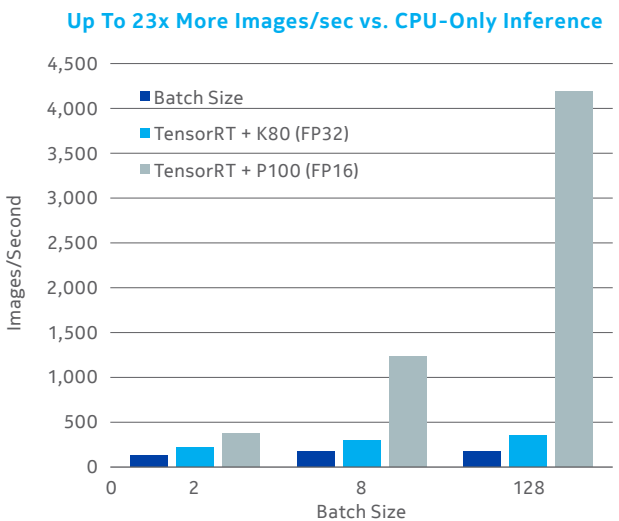
Another performance comparison is presented in **Figure 4**. The attentive reader will notice, that the better results are achieved with the reduced floating-point precision (FP16 instead of FP32). While the main conclusion of scientific research publications on suitability of reduced precision computations for ML/AI is that precision reduction accelerates computations while not affecting classifier quality, and such approach is justified, it leaves us with a question on performance of the old hardware in such conditions.

Figure 3 Deep learning inference troughput comparison *(nVidia 2017)*



Workload: ResNet-50 | CPU: 1X XeonE5-2690v4 @ 2.6 GHz | GPU: Add 1X Tesla P100 or V100

Figure 4 GoogLeNet throughput comparison for different hardware *(Allison Gray 2017)*



GoogLeNet, Tesla P100 + TensorRT (FP16), Tesla K80 + TensorRT (FP32), CPU-Only +Caffe (FP32)
CPU: 1 Socket Broadwell E5-2690 v4@2.6GHz with HT off

5. Example program: adding two arrays

This sample code (based on (Harris 2017)) presents the usual way to perform a parallel operation, as of CUDA version 9.x. Apart from compilation with a specific compiler, such as *nvcc* for CUDA, main differences relate to parallel execution (calculation of *index* and *stride* in the *add* function, calculation of block size and the number of blocks before the "kernel" call), but there are some CUDA-specific changes: memory management (**cudaMallocManaged()** in place of **malloc()**, **cudaFree()** instead of **free()**), and additional synchronization between GPU processing and main program (**cudaDeviceSynchronize()**).

Listing 1

```
#include <iostream>
#include <math.h>

// Kernel function to add the elements of two arrays
__global__
void add(int n, float *x, float *y)
{
  // Calculate region
  int index = blockIdx.x * blockDim.x + threadIdx.x;
  int stride = blockDim.x * gridDim.x;
  for (int i = index; i < n; i += stride)
    y[i] = x[i] + y[i];
}

int main(void)
{
  int N = 1<<20;
  float *x, *y;

  // Allocate Unified Memory – accessible from CPU or GPU
  cudaMallocManaged(&x, N*sizeof(float));
  cudaMallocManaged(&y, N*sizeof(float));

  // initialize x and y arrays on the host
  for (int i = 0; i < N; i++) {
    x[i] = 1.0f;
    y[i] = 2.0f;
  }

  // Run kernel on 1M elements on the GPU
  int blockSize = 256;
  int numBlocks = (N + blockSize - 1) / blockSize;
  add<<<numBlocks, blockSize>>>(N, x, y);

  // Wait for GPU to finish before accessing on host
  cudaDeviceSynchronize();

  // Check for errors (all values should be 3.0f)
  float maxError = 0.0f;
  for (int i = 0; i < N; i++)
    maxError = fmax(maxError, fabs(y[i]-3.0f));
  std::cout << "Max error: " << maxError << std::endl;

  // Free memory
  cudaFree(x);
  cudaFree(y);

  return 0;
}
```

Before you try this at home – please pay attention to your GPGPU card model and available libraries or toolkits. There are differences between capabilities of different GPGPU chipsets and different versions of supporting software. For example, CUDA 9 doesn't support the first generation of Tesla cards, also ray tracing units of recent GPGPUs are not supported. While cross-compilation is possible, there is no indication that code is not operational (sometimes results are wrong, sometimes the program crashes with memory access violations). Some issues related with parallel computing are addressed by external libraries, like Hemi (Harris, Hemi – Portable CUDA C/C++ (GitHub repository) 2015).

6. The future?

Compared to cryptocurrency mining development, it's expected that GPGPU is not the last word in the matter of computation power for AI and ML. While the most common association for Bitcoin is GPU, it was not the last step of the journey. In the search for faster computation, FPGA (Field-programmable Gate Array) chips were utilized. FPGA chips consist of an array of programmable logic blocks and reconfigurable interconnections between them. In contrast to a general CPU, where more sophisticated functions must be expressed as the result of several arithmetic and logic operations performed on step-by-step basis (program execution), FPGA chips are "wired" to perform a specific operation, taking advantage of their internal block structure (usually a look-up table, adder, and latch/D flip-flop) and interconnections hierarchy. After a configuration step (which takes some time), they perform the given function. Any change of the programmed function requires chip reconfiguration. While in the beginning FPGA configuration times were quite long, recent development allowed partial chip reconfiguration, thus enabling on the fly configuration according to the actual needs. Coupling ARM CPU cores with FPGA into single chip (SoC) boosted embedded systems development (lets name Xilinx Zynq SoC family), so we may expect that FPGA-enabled server grade CPUs may do the same to the ML/AI domain, allowing implementations to be made in hardware.

References

[1] Allison Gray, Chris Gottbrath, Ryan Olson and Shashank Prasanna. 2017. *Deploying Deep Neural Networks with NVIDIA TensorRT.* 04 02. https://devblogs.nvidia.com/deploying-deep-learning-nvidia-tensorrt/.

[2] Harris, Mark. 2017. *An Even Easier Introduction to CUDA.* 01 25. https://devblogs.nvidia.com/even-easier-introduction-cuda/.

[3] —. 2015. *Hemi - Portable CUDA C/C++ (GitHub repository).* http://harrism.github.io/hemi/.

[4] Lazorenko, Andriy. 2017. *TensorFlow performance test: CPU VS GPU.* 12 27. https://medium.com/@andriylazorenko/tensorflow-performance-test-cpu-vs-gpu-79fcd39170c.

[5] nVidia. 2017. *nVidia Tesla V100 GPU Architecture. The World's Most Advanced Data Center GPU.* 08. http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

[6] —. 2017. *Tesla V100 Data Center GPU.* Accessed 10 10, 2018. https://www.nvidia.com/en-us/data-center/tesla-v100/.

[7] P. Y. Simard, D. Steinkrau and I. Buck. 2005. "Using GPUs for Machine Learning Algorithms." *Eighth International Conference on Document Analysis and Recognition (ICDAR'05).* Korea: IEEE Computer Society. 1115-1119.

[8] Wikipedia. n.d. *Wikipedia.* Accessed 10 10, 2018. https://en.wikipedia.org.

About the author

I have extensive experience in systems and networks administration, including GPU-based systems used for scientific computations. My interest in machine learning was reflected in my master's thesis and a few publications. In Nokia I am a member of Site Reliability Engineering Team at GS CS Network Engineering.

Wojciech Penar
DevOps Engineer
GS CS Network Engineering

What is Apache Spark?

Tomasz Szandała
Software Configuration Engineer
MN CDS Software Configuration Management

NOKIA

Apache Spark is an open-source distributed computing framework that was initially developed at UC Berkeley by PhD student Matei Zaharia. As compared to the Google's disk-based, two-stage MapReduce of Hadoop, Spark provides up to 100 times faster performance for a few applications with in-memory primitives. This makes it convenient for machine learning algorithms, as it allows programs to load data into the cluster's memory and query for the data constantly. The Spark project consists of various components such as Spark Core and Resilient Distributed Datasets (called RDD), Spark SQL, Spark Streaming, Mllib (Machine Learning Library) and GraphX - tool for graph computations.

Scala is a multi-paradigm programming language working on Java Virtual Machine[8]. Although it runs on JVM and it is mostly compatible with Java, most concepts are taken from other languages like Haskell or Erlang. It has been designed for expressing general programming patterns in an elegant, precise and type-safe way. One of the prime features is that it integrates the features of both object-oriented and functional languages quite smoothly. It is a pure object-oriented language, as every value in it is an object. The Objects' behavior and types are achieved through classes and traits that resemble Java interfaces. Scala is also a functional language, as every function in it is a value. By providing a lightweight syntax for defining anonymous functions it provides support for higher-order functions.

Luckily, we do not have to learn Scala to use Apache Spark. Only Spark Core is Scala-only, all other components provide tooling in both Scala and Python with API for other, non-canonical languages.

1. At the beginning there was Hadoop

Long before Matei started working on Spark everyone involved in distributed processing were using Hadoop and its MapReduce. MapReduce used in Hadoop is not perfect for many reasons, e.g. it is not a good choice when it comes to real-time processing. It is batch-oriented, hence it is executed as periodic jobs that take time to process the data and provide results. It takes minutes to complete a job, which mainly depends on the data amount and number of nodes in the cluster. This is useless in processing of streamed data, where we need real time answer.

2. Spark Streaming

Spark streaming is one of main parts of Spark API's extension that allows high-throughput, scalable, and fault-tolerant stream processing of data streams that are live. There are many sources of constant sources of data such as ZeroMQ,RabbitMQ, Twitter, Instagram, stock markets or Kafka. The data is read in real time from the source and once it is processed, you can push it to databases, data warehouses or live dashboards.

First Spark takes streamed data input and divides them into batches. After this, the Spark engine processes those streams and generates the final stream results in batches.

The most noticeable feature of Spark Streaming is Discretized Stream (DStream) – the basic abstraction provided by Spark Streaming. It is a continuous stream of data, that is received from a data source or a processed data stream generated by transforming the input stream. Internally, a DStream is represented by a continuous series of RDDs and each RDD contains data from a certain interval of continuous stream.

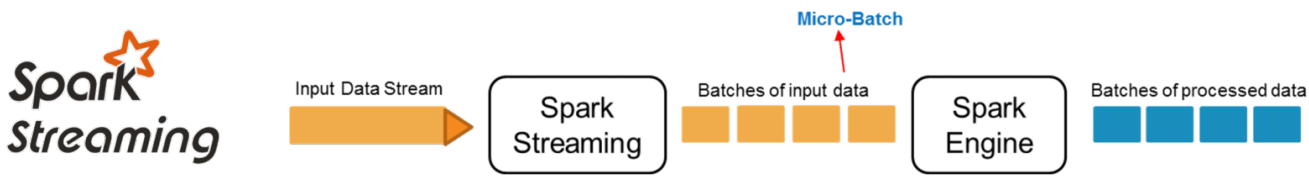
What is RDD?

RDD (Resilient Distributed Datasets) in a Spark idea for representing data. Formally they are a read-only, partitioned collection of records that provides a convenient API. Collection can be text / binary files, json, xml or etc. data.

- RDD provide a convenient solution for processing large datasets on cluster computing frameworks such as MapReduce by addressing some key issues:
- data is kept in memory to reduce disk I/O; this is particularly relevant for iterative computations - not having to persist intermediate data to disk
- fault-tolerance (resilience) is obtained not by replicating data but by keeping track of all transformations applied to the initial dataset (the *lineage*). This way, in case of failure lost data can always be recomputed from its lineage and avoiding data replication again reduces storage overhead
- lazy evaluation – actions on RDD contents are carried out first when they are needed

Itis worth to mention that RDDs are immutable (read-only), this fulfills functional programming paradigm of immutable variables.

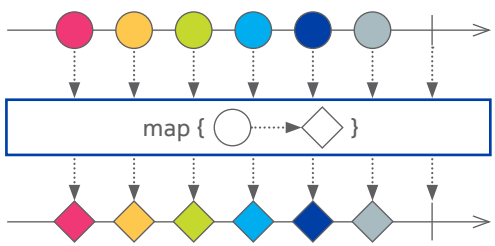
Figure 1 Spark Streaming concept diagram

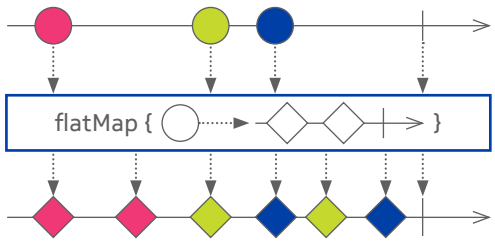


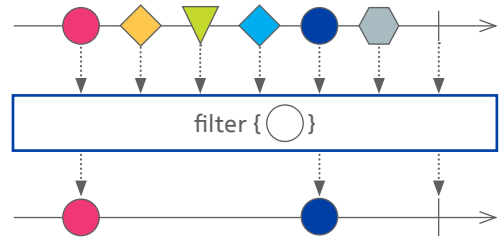
Spark recommends using functional paradigms for writing its scripts. In functional code, the output value of a function depends only on arguments that are passed to the function, so calling a function *f* twice with the same value for an argument *x* produces the same result *f(x)* each time. Eliminating side effects, i.e., function output depends only on the input data and does not depend on the function context, can make it much easier to understand and predict the behavior of a program, which is one of the key motivations for the development of functional programming. Furthermore, all variables are immutable, if we name something 'X' we cannot assign other value. Like in math: we cannot have once x=5, other time x=3 in the same exercise. This behavior gives even more benefits in parallel processing, where we can safely call functions on independent nodes.

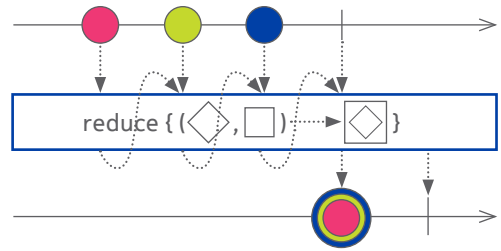
Table 1 The following are some of the popular transformations on DStreams - also typical methods found in all functional languages

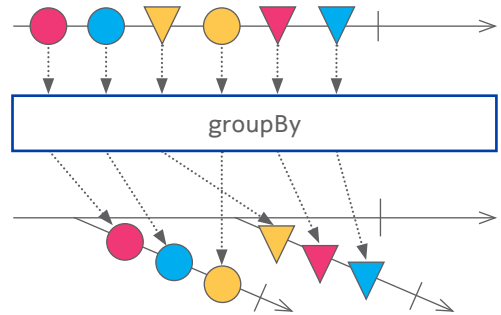
map(func)	map(func) returns a new DStream by passing each element of the source DStream through a function func.
flatMap(func)	flatMap(func) is similar to map(func) but each input item can be mapped to 0 or more output items by unwrapping (flattening) passed object if it is a container.
filter(func)	filter(func) returns a new DStream by selecting only the records of the source DStream on which func returns true.
reduce(func)	reduce(func) returns a new DStream of single-element RDDs by aggregating (merging) the elements in each RDD of the source DStream using a function func.
groupBy(func)	groupBy(func) returns the new RDD which basically is made up with a key and corresponding list of items of that group.











3. Twitter Sentiment Analysis using Apache Spark Streaming

Now that we have understood the core concepts of Spark Streaming, let us solve a real-life problem using Spark Streaming. The problem: design a Twitter Sentiment Analysis System where we populate real-time sentiments for crisis management, service adjusting and target marketing.

Sample applications of sentiment analysis:

- Predict the success of a movie
- Predict political campaign success
- Decide whether to invest in a certain company
- Targeted advertising
- Review products and services

For this application I have chosen Python due to simplicity and popularity across data scientist. Python is great for data science modeling thanks to its numerous modules and packages that help achieve data science goals. What it was missing was a solution to better parallelize the computations. That's where the Apache Spark comes in.

One of the first requirements is to get access to the streaming data; in this case, real-time tweets. Twitter provides a very convenient API to fetch tweets in a streaming manner

One more thing to note is that I will work in local mode with my notebook. The local node is often used for prototyping, development, debugging, and testing. However, as Spark's local mode is fully compatible with the cluster mode, codes written locally can be run on a cluster with just a few additional steps.

Let's write the sample application, in which we will be processing streaming data from Twitter in real time. One of the first requirements is to get access to the data, to do so one needs to register on Twitter website an acquire personal credentials and keys to fetch messages, using a very convenient API, in a streaming manner. In addition, we will also be needing Kafka database to store the messages before actual processing. Kafka provides a distributed queuing service which can be used to buffer the data when the data creation rate is higher than processing rate.

In order to fetch tweets from twitter streaming API and save them to Kafka queue, we have the python script twitter_2_kafka.py. The script will need mentioned twitter authentication tokens. Once we have the authentication key we can start downloading tweets from the twitter stream API and push them to the twitter stream topic in Kafka. Note that some of the imports are external python libraries. They are easily installable on the one machine, but if you are running multinode Apache Spark cluster, you have to make sure that these libraries are installed on all of the worker nodes.

Listing 1 Script for fetching data from Tweeter and saving them into Kafka database

```
from kafka import SimpleProducer, KafkaClient
import tweepy


class TweeterStreamListener(tweepy.StreamListener):

    def __init__(self, api):
        self.api = api
        super(tweepy.StreamListener, self).__init__()
        # default port for Kafka is 9092
        client = KafkaClient("localhost:9092")
        self.producer = SimpleProducer(client, async = True,
                           batch_send_every_n = 1000,
                           batch_send_every_t = 10)

    def on_status(self, status):
        msg = status.text.encode('utf-8')
        try:
            self.producer.send_messages('spark_stream', msg)
        except Exception as e:
            print(e)
            return False
        return True

    def on_error(self, status_code):
        print("Error received in kafka producer")
        return True # Don't kill the stream

if __name__ == '__main__':

    consumer_key = 'Your consumerKey from Twitter'
    consumer_secret = 'Your consumerSecret from Twitter'
    access_key = 'Your accessToken from Twitter'
    access_secret = 'Your accessTokenSecret from Twitter'

    auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_key, access_secret)
    api = tweepy.API(auth)

    # Create stream and bind the listener to it
    stream = tweepy.Stream(auth, listener =
    TweeterStreamListener(api))

    # Custom Filter rules pull all traffic for those filters in
      real time.
    stream.filter(track = ['love', 'hate'], languages = ['en'])
```

In the above code we have one class - we want to overwrite some methods of standard Twitters StreamListener like on_status - method called when there is change in stream or on_error to not interrupt the process when there is an incorrect data from Twitter service. This script initializes Twitter API connection and begins listening for all messages that contain words 'love' or 'hate'.

The next step computes total count of the number of positive and negative words that have been found in each message. The word-lists for positive words and negative words are given in the respective positive.txt and negative.txt files, but we can use other sources, like Python's module AFFIN.

Listing 2 Script using Apache Spark Stream to classify sentiment analysis of collected messages

```
from pyspark import SparkConf, SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils

def main():
    conf = SparkConf().setMaster("local[2]").setAppName("Streamer")
    sc = SparkContext(conf=conf)
    # Streaming context interval of 10 seconds
    ssc = StreamingContext(sc, 10)
    ssc.checkpoint("checkpoint")

    pwords = set(line.strip() for line in open("positive.txt",'r'))
    nwords = set(line.strip() for line in open("negative.txt",'r'))

    counts = stream(ssc, pwords, nwords, 100)

def orientation(word,pwords,nwords):
    if word in pwords:
        return ('positive',1)
    elif word in nwords:
        return ('negative',1)

def updateFunction(newValues, runningCount):
    if runningCount is None:
        runningCount = 0
    return sum(newValues, runningCount)

def stream(ssc, pwords, nwords, duration):

    kstream = KafkaUtils.createDirectStream(
        ssc, topics = ['spark_stream'], kafkaParams = {"metadata.
broker.list": 'localhost:9092'})
    raw_tweets = kstream.map(lambda x: x[1].encode("utf-8"))

    # use Spark's map method for pre-processing
    tweets = pre_process_messages(raw_tweets)

    words = tweets.map(lambda k: k.split(" "))
    wordsType = words.map(lambda k: orientation(k,pwords,nwords))
    ct = wordsType.reduceByKey(lambda a, b : a + b)

    counts = []
    cts = ct.updateStateByKey(updateFunction)
    ct.foreachRDD(lambda t,rdd: counts.append(rdd.collect()))
    cts.pprint()

    ssc.start()
    ssc.awaitTerminationOrTimeout(duration)
    ssc.stop(stopGraceFully=True)

    return counts

if __name__=="__main__":
    main()
```

In the second snippet we are actually processing received messages. First, we initialize the Spark Context - the core of Spark streaming, to use it in connection to Kafka's queue from which we will continuously (or rather every 10 seconds) receive messages. Tweets saved in Kafka are in "raw" format, which means emoticons, punctuation marks or even typos. In order to get rid of this "noise" we have to perform pre-processing of messages that unifies contents for easier classification. The more advanced words pre-processing include stemming and lemmatization. The pre-processing also changes all words to lowercase, to reduce the amount of possible variations.

Such prepared lists of words are finally processed using positive and negative lists. We compute how many words are positive and how many negative. In the end we summarize the score and assign label for each tweet: POSITIVE or NEGATIVE. In this example we ignore NEUTRAL results. In the variable 'counts' we have amount of both message types from given time window. We can present them e.g. on a sentiment trend graph.

Where lies the power of Spark? In methods like 'map' used here few times. Because this function processes each element independently it can be easily distributed on few computing nodes to reduce processing times. Of course, we could use more advanced sentiment analysis, like Support Vector Machine, deep neural networks or Naive Bayes, but this is not the topic of this paper.

4. Real life Spark Applications

At the beginning Spark Streaming was a fancy tool for data scientist to do some scientific research, but in recent years it increased value of data analysis for literally every company. Various businesses can now process stream data from real-time application, e.g. auction platform, leading to more accurate targeting of displayed advertising or better consumer engagement and higher conversion. Any telecommunications provider can collect metrics from millions of mobile phones and analyze them to determine where to place new cell phone towers and upgrade aging infrastructure, resulting in improved service quality. In Information Technology we can use Spark for log processing to detect unusual events occurring in streams of data, enhancing IT teams to take preventive action before service quality fully degrades.

5. Summary

Apache Spark is a general-purpose framework for large-scale distributed data processing. It supports rapid parallel processing of big data and allows for code reuse across batch, interactive and streaming applications. Spark outperforms classical MapReduce and comes with even more tools, like Streaming feature described above. Currently Apache Spark is officially used by over 150 global companies including NOKIA for not only data processing, but also multiple other tasks that might benefit from distributed processing. Spark features are not limited to the data handling, it can even work as Continuous Integration tool but this is a material for another paper.

References

[1] https://spark.apache.org/docs/2.2.0/quick-start.html
[2] https://www.janbasktraining.com/blog/what-is-spark/
[3] Tomasz Szandała, Internet rzeczy – wyzwania dla naukowców, Creativetime, 2017
[4] Anuj Saxena, How to Use the Kafka Streams API, Big Data Zone, 2017
[5] Tathagata Das, Why companies like Uber and Netflix are adopting Spark Streaming to handle big and fast data challenges, Datanami, 2015
[6] https://spark.apache.org/powered-by.html
[7] Yu Peng, Andrew Chen and Prakash Chockalingam, Continuous Integration & Continuous Delivery with Databricks, 2017
[8] The Origins of Scala, Bill Venners, Frank Sommers, 2009

About the author

I am PhD student at Wroclaw University of Science and Technology in the field of Artificial Intelligence. Also I am member of Creation & Development Group KREDEK – university's science club. In NOKIA I work as Software Configuration Engineer in Wroclaw's DevOps Team. My daily work consists of deploying and maintaining few hundreds hosts for multiple purposes: Jenkins, Web Applications, compilation servers, etc. My favorite task in life is solving problems since with computers impossible is nothing, it just takes a bit more time.

Tomasz Szandała
Software Configuration Engineer
MN CDS Software Configuration Management

Analytics Ecosystem with Nokia AVA

Piotr Godziewski
Head of Analytics Engine
GS Customer Support

Paweł Ślawski
Technical Leader, AVA Platform
GS CS Analytics Engine

NOKIA

1. Motivation for in-house analytics platform

The ambition to use machine learning for revenue recognition and operational efficiencies requires the teams to look for the enablers which allow professional productization of the AI applications. Whether it is a customer-facing service solution or an internal automation tool, they all need a reliable development, deployment and delivery platform.

In the cloud native world, this is handled by the Infrastructure-as-a-Service and Platform-as-a-Service providers. Many companies decide to offload their infrastructure and platform operations and fully rely on the off-the-shelf solutions from Amazon, Microsoft, Google and others.

Nokia however, the leading telecom software vendor, has a very deep understanding of cloud native development as part of its product and service portfolio, with Network Function Virtualization being one example [1]. At the same time, the telco market remains quite challenging. Flat or declining ARPU (Average Revenue Per User) forces the operators to look for the operational savings to compensate.

With the customer networks becoming ultra-dense and more heterogenous than ever before, with the introduction of 5G and IoT, the new user experience is being defined by the higher throughputs and the lower latency across all network segments. The combination of the skillset and experience with the unique market and customer characteristics, justifies the decision to build an in-house analytics platform. This platform addressed the specific requirements of telecom industry, both mobile and fixed networks, as well as the vertical opportunities such as healthcare, IoT, smart cities or enterprises.

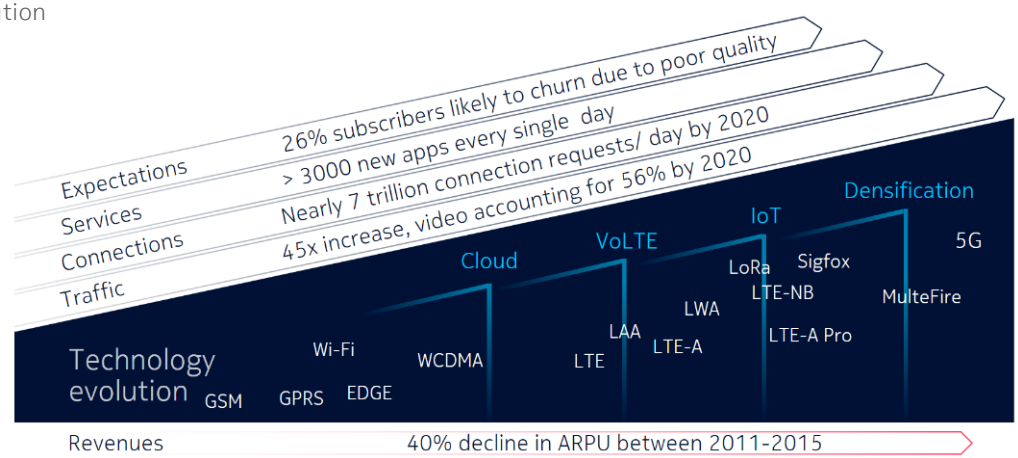
The foundations of the platform were provided by the development of the automation engines for services such as network planning, optimization and care. It did not take long before we realized that we could focus on delivering the new innovative and data-centric solutions, rather than improving processes gradually through automation. Soon after it evolved into an analytics ecosystem blended with extreme automation that creates environment for the unique customer solutions as well as the internal services.

Today Nokia AVA – Analytics Virtualization Automation – is the go-to platform for the service solutions which involve data collection, data engineering, machine learning and business intelligence.

2. Platform development

The development highly relies on open source and it works in both directions. While the AVA software engineers reuse the open source components, they contribute back to the community with new features, bug fixes, improvement ideas. This approach has a direct impact on the chosen technology stack. Apache Spark is at the heart of the AVA analytics engine to allow large-scale distributed data processing [2]. Read more in Apache Spark chapter of this book [3]. Containerized machine learning applications and microservices for production deliveries are deployed with Kubernetes or Apache Mesos depending on the use case. Data science teams can experiment and test the machine learning projects in Apache Zeppelin [4], in an interactive way before wrapping the final design in the production containers. Our presence in the open community goes beyond single components. By following the Open Service Broker API project, we are continuously looking into infrastructure and platform services in which others previously excelled more than us.

Figure 1 Telco Evolution



All this is now serving solution architects, data engineers, data scientists and software teams developing the machine learning applications for revenue and operations. The use case factory (think of a tribe in Spotify culture,[5] is responsible for developing and operating their production-grade analytics services on AVA.

Having AVA platform R&D in-house makes it easier to onboard new users from variety of less formal machine learning guilds across the company, usually concentrated around R&D sites or ambition to drive specific use cases.

3. Data

The variety of analytics applications is being developed and deployed on AVA and they all rely on various data types. These may include:

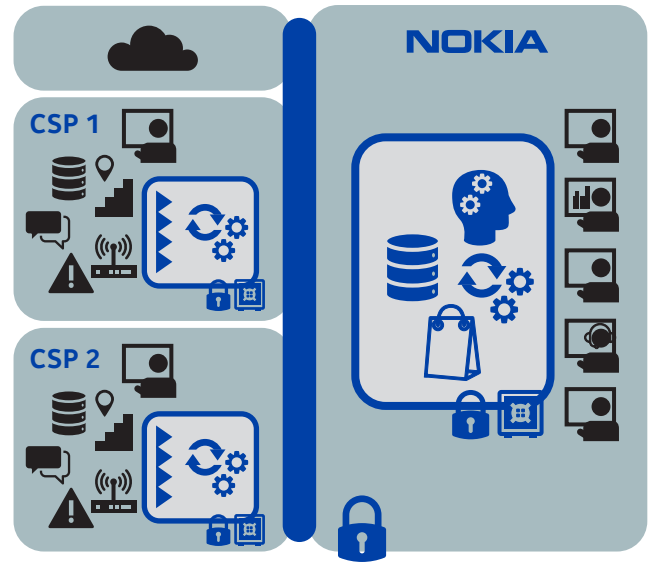
- enterprise data; e.g. sales opportunity funnel, contract management data,
- customer care data; e.g. support tickets, software bugs, troubleshooting logs,
- and the most commonly used, data from network products deployed in the field, or under verification in the test laboratories.

It is therefore one of the key AVA design criteria to support the analytics community with the data collection mechanisms. This allows the engineers to onboard the data sets in a trusted and secured way following the data privacy regulations. While the enterprise and customer care data is typically accessible from the APIs of the corporate IT systems, the variety of telecom products (think multi-vendor, not only Nokia) presents more complex implementation problem.

AVA R&D continuously develops the common architecture runway with the internal product units. The optimum business logic and functionality split must be in place between the network element (or its management system) and the AVA data collectors. Only then the performance and security criteria can be met.

Virtualization and cloudification of the telecom products made it possible to move more network intelligence to the edge, closer to the subscriber for lower latencies and improved responsiveness to service degradation. ReefShark is the Nokia chipset technology boosting the base station AI capabilities [6]. Deep learning is now within computational range for radio and baseband units which means that the time and service critical analytics are executed on-premise using the product infrastructure. Not everything has to be performed in the remote vendor-controlled site.

Figure 2 The business logic split of the edge (customer) and vendor environment



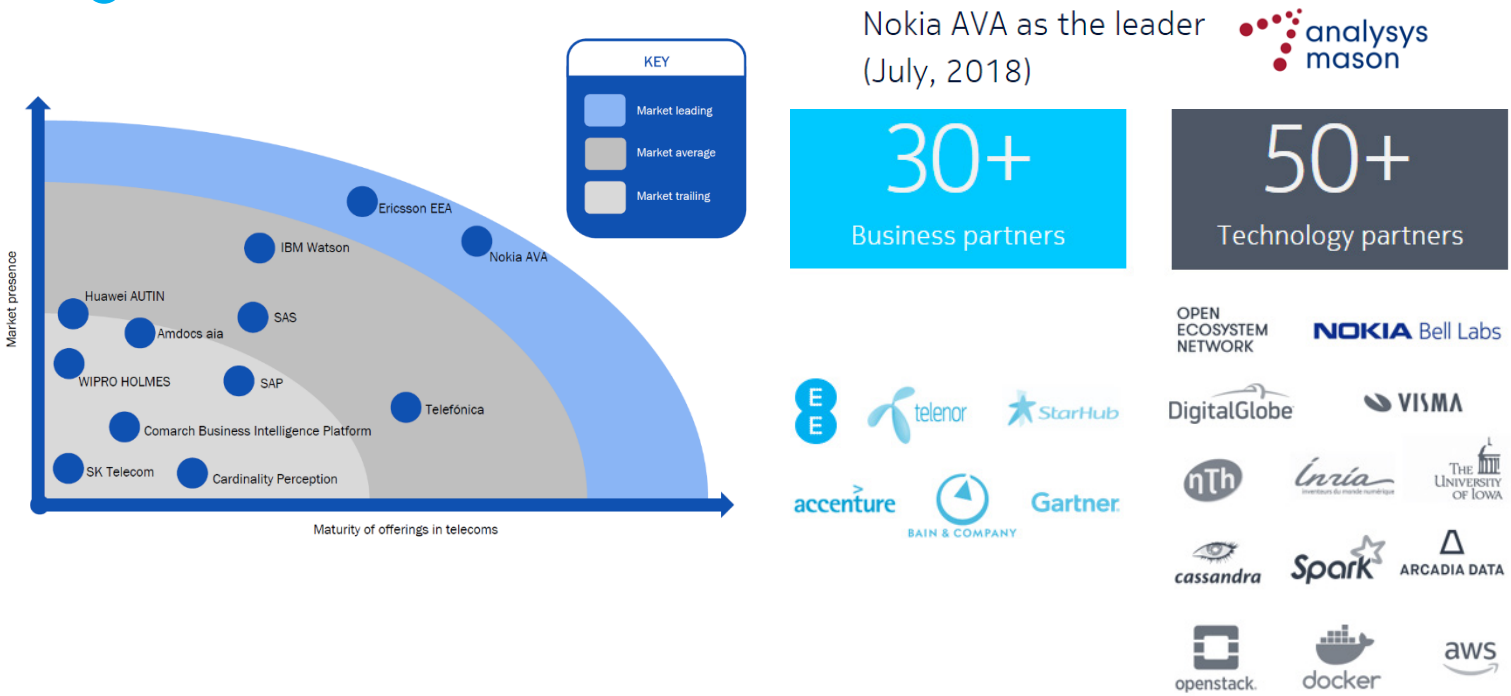
It is therefore essential for the architecture teams to agree which data (and how) leaves the customer premises for further processing, customer care and solution development. The AVA platform cloud native design makes it deployable on the very same product infrastructure if a customer desires to keep the full control of the analytics and automation environment. This is usually dictated by the data in use and the corresponding regulatory laws.

4. Market leader

According to Analysys Mason, the global consulting and research company specializing in telecom, media and technology, Nokia AVA is the market leader and the most mature analytics solution in the industry [7]. The telco analytics built on top of the AVA platform forms a unique service solution offering.

Completely new data use cases, with analytics and machine learning, now find their way from experimentation, through development to production. AVA became the strategic enabler for co-development of these use cases along with the operators.

Figure 3 Nokia AVA - Market Leader [7]



4.1 Mobility Analytics

Mobility Analytics [8] is an example of co-development effort with Starhub, a Singapore-based telco operator. Network data and subscriber location information are transformed into new services and revenue streams for the operator ultimately creating value for the smart city digital players [9].

This solution utilizes a variety of subscriber's mobility parameters such as position, velocity and movement direction. Billions of MDT (Minimization of Drive Test) measurements collected from the devices are used to train the Bell Labs designed neural network to find patterns and the best performing areas. Next, it applies the proven network parameter settings to the areas of poor performance and optimizes the subscriber's experience, i.e. less service interruptions (reduced dropped calls) and higher mobile broadband speed (increased throughput).

Mobility Analytics has been enabled by AVA.

Figure 4 Mobility Analytics

4.2 Ticket Handling Automation

Ticket Handling Automation represents an internal ambition to augment customer care engineers with machine learning and extreme automation. It connects data, business units and teams between various phases of customer ticket lifecycle to eliminate repetitive human intervention. Metadata-based ticket segmentation is only an initial step which takes us further to the target of similar ticket recognition. The advantage over third party solutions comes from utilizing multiple internal knowledge repositories which hold keys to resolving most common customer issues (e.g. wrong parameter settings, non-optimum feature sets, missing software patches). Deeper troubleshooting requires detailed ticket description and network element logs, that represents complex natural language and text processing.

To identify a root cause and recommend a solution, significant amount of data reflecting patterns in network behavior is needed. Various business units work together inside AVA ecosystem collecting terabytes of data, preparing training sets and finding the most optimum neural network parameters. At the same time, AVA platform teams introduce the required technology enablers (e.g. storage and pipeline services) to assure a reliable production environment.

Again, AVA enables this system to be secure, reliable and scalable enough to handle customer ticketing loads.

5. Data science experience and development to production

AVA serves both the research and development teams in delivering end-to-end analytics solutions. It enables the data engineers and scientists as well as the software developers to co-exist and share the same ecosystem.

Data scientists can setup their own analytics lab for experiments with just a few clicks. The data flows, storage, distributed computing engine, dashboarding are all securely provided for immediate use. Python, Scala, R and other programming runtimes await in the interactive notebooks. A distributed computing engine in the cloud environment deals with processing the experiments. One can share and collaborate with the entire analytics community. All that stimulates innovation and creativity and serves as a sandbox for:

- Data analysis
- Data extraction, cleaning, transformation, preparation
- Data visualization
- Machine learning experiments
- Rapid application prototyping

Extremely short cycle time is what matters in the experiments but when the product is ready, the software engineering starts to productize it. Only high-quality and production-ready analytics content qualifies to be containerized and deployed. It is not uncommon to throw away a prototype and start the application design according to the customer requirements. Data flows are being optimized and automated to run as spark jobs when data arrives. The data visualization teams work out the best user experience, no matter internal or external customer.

Ultimately AVA connects the world of research and experimentation (typically occupied by the data scientist) with the rigorous rules of production deliveries (handled by the software engineers).

6. Demand for new skills

The analytics solutions developed with AVA creates new customer solutions and bring greater automation in the internal processes. The revenue and operational benefits are the key drivers. However, there is one more aspect not so noticeable in the profit and loss sheets. The use of platforms such as AVA leads to the major change in the corporate tooling landscape and generates the demand for new competence profiles.

There are couple of disadvantages with the standalone purpose-built tools, which we used to rely on when delivering the data-centric customer services:

- Monolith software architecture prevents the development teams from fully iterative feature delivery in response to the changing customer requirements.
- All-in-all tools have tendency to generate an unnecessary internal competition instead of encouraging software reuse and the inner sourcing practices.
- The increasing amount of business and functional logic makes such tools very hard to maintain.

A platform such as AVA shifts the focus of the analytics teams to the actual experiments, rapid prototyping and eventually providing the production-grade applications without the need to build the common enablers again and again (e.g. security, reliability, authentication).

The AVA platform does not lock its community with one analytics framework, one programming language or a database solution. There are certain ground rules for development and deployment, but as much flexibility as possible is left to the teams for experimentation. It stimulates onboarding of:

- Machine learning experts
- Data engineers

- Software engineers
- Solution architects

The telecom services have always been driven by the data (network simulations, dimensioning, planning, optimization, quality assurance, etc.). This experience is a perfect foundation for many teams stepping in to the AVA ecosystem and the analytics solutioning.

References

[1] Network Function Virtualization (NFV), https://networks.nokia.com/solutions/network-functions-virtualization-nfv
[2] Apache Spark, https://spark.apache.org/
[3] **Nokia** Machine Learning in Telecommunication Networks is Happening Now. Check how the experts do it., Apache Spark
[4] Apache Zeppelin, https://zeppelin.apache.org/
[5] Spotify engineering culture, https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1/
[6] Nokia ReefShark, https://networks.nokia.com/5g/reefshark
[7] Analysys Mason, "Telecoms AI ecosystems in digital transformation: increasing the level of automation in critical processes" - AVA positioned as a market leader http://www.analysysmason.com/Research/Content/Short-reports/tae-automation-report-rma14/#13%20July%202018
[8] Nokia Mobility Analytics, https://networks.nokia.com/services/mobility-analytics
[9] Nokia and Starhub co-develop analytics services for digital cities, https://www.nokia.com/en_int/news/releases/2018/02/20/nokia-and-starhub-co-develop-analytics-services-for-digital-cities

About the authors

Piotr Godziewski leads the Analytics Engine tribe, the R&D unit responsible for the AVA platform and data collection software. He was a part of an interdisciplinary team under CTO developing the Nokia analytics strategy. His professional involvement in the data analytics includes the automation solutions for the global and market sales units. He is a big fan of Andrew Ng's work and his contribution in the deep learning development.

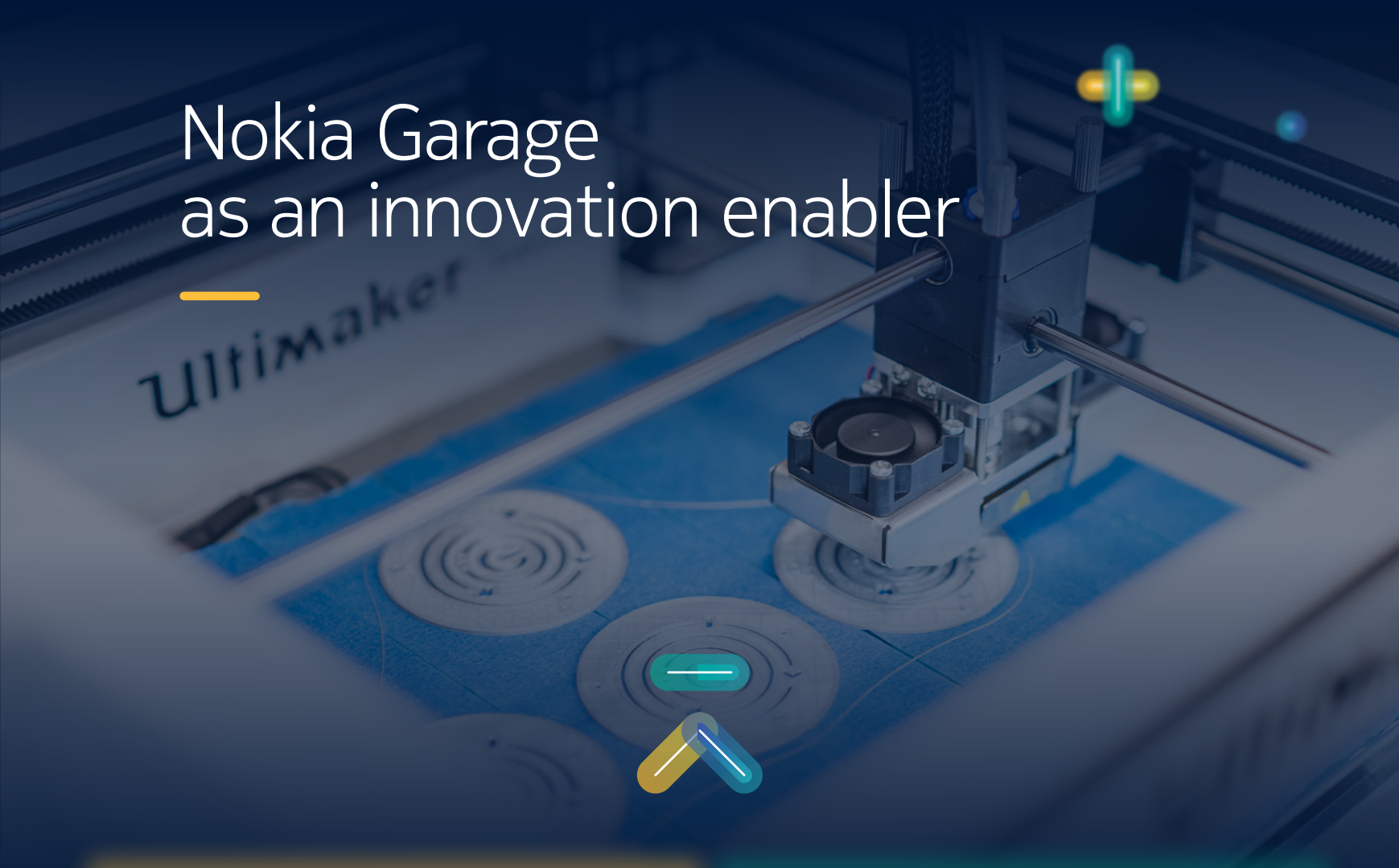
Paweł Ślawski is a Technical Leader in Analytics Engine tribe. He isResponsible for implementation of Nokia AVA Platform and Analytics Use Cases. Leads Mobility Analytics project and analytics use cases for Nokia like Ticket Handling Automation. Nokia Wrocław Machine Learning Guild member.

NOKIA

GARAGE

LIGHT ON
THE MAGIC
OF TECHNOLOGY

Nokia Garage as an innovation enabler



Nokia Garage is a place, where business, technology and knowledge meet, and innovators are supported with the best of Nokia experts.
It is the place which is open for anyone (both employees and external people), who has an idea and wants to transform it into a project, product or process with Nokia Garage Community.

NOKIA GARAGE COMMUNITY MEMBERS

- …Share knowledge
- Think out of the box
- Build networking
- Collaborate
- Have fun…

Nokia Garage in a nutshell

PEOPLE – Engaged community built with Nokia experts, scientists, students, technology enthusiasts and business partners.

PLACE – About 500 square meters space designed with innovators and for innovative work. Nokia Garage offers three zones for different purposes:

- Creative zone for brainstorming's, workshops, customer visits
- Makerspace for fast prototyping
- Lab for testing solutions

TECHNOLOGY – Access to the most advanced technologies from Nokia to create innovative solutions based on: 5G, IoT, ML, AI and EdgeCloud.

Cutting edge projects

GPU-powered ML servers:

- 8 cores CPU's
- GTX1080 GPU's
- 64 GB DDR4 memory
- 500 GB SSD drives
- NVIDIA Docker deployment
- 24/7 remote and on-site availability for Nokia Garage Ecosystem Projects

Main purpose:

- AI / Machine / Deep Learning
- Video and graphics data processing / 3D rendering
- Modeling and simulations
- … and many more

Contact

West Link building
ul. Szybowcowa 2
54-130 Wroclaw

https://facebook.com/nokiagarage
https://www.instagram.com/nokiagarage
https://nokiagarage.pl
hello@nokiagarage.pl

Strzegomska Street 36
53-611 Wrocław
Reception 1st Floor, Green Towers B
Opening hours 8:00 – 18:00
Phone: +48 71 777 3800
Fax: +48 71 777 30 30

Lotnicza Street 12
54-155 Wrocław
Reception Ground Floor, West Gate
Opening Hours 8:00 – 18:00
Phone: +48 71 777 4002
+48 71 777 4001

Szybowcowa Street 2
54-130 Wrocław
Reception Ground Floor, West Link
Opening hours 8:00 – 18:00
Phone: +48 71 777 4000

e-mail: kontakt@nokiawroclaw.pl
www.nokiawroclaw.pl