Design of a Graph Neural Network

to predict the optimal resolution of the Sonar Performance Model.

Jakub Pietrak

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Design of a Graph Neural Network

to predict the optimal resolution of the Sonar Performance Model.

Master of Science Thesis

For the degree of Master of Science in Mechanical Engineering at Delft University of Technology

Jakub Pietrak

September 10, 2020

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by TNO.
Their cooperation is hereby gratefully acknowledged.

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Design of a
Graph Neural Network

by

Jakub Pietrak

in partial fulfillment of the requirements for the degree of

Master of Science Mechanical Engineering

Dated: September 10, 2020

Supervisor(s):

Prof. Dr. Em. Bart De Schutter

Dr. ir. Joris Sijs

Dr. ir. Jens Kober

Reader(s):

Abstract

Graph Neural Networks are a unique type of Deep Learning models that have a capability to exploit an explicitly stated structure of data representation. By design they carry a strong *relational inductive bias*, which is a set of assumptions that makes the algorithm prioritize some solutions over another, independent of observed data. This makes the method especially interesting for applications to problems, that are naturally relation-centric, or in which local interactions between features are the main value of interest.

The presented research case, aims to explore GNN potential in application to an Ocean Acoustics problem. Using the geometric ray-tracing algorithm, BELLHOP, a large number of underwater sound propagation scenarios was simulated. Each scenario is described by a limited set of parameters and a Sound Speed Profile function. The latter, acting as a guideline for estimating paths of rays travelling through a water column, has a critical impact on sound propagation mode. For the data-driven model to effectively capture the acoustic phenomena, requires a mean of representing interactions in very scarce feature space and especially with respect to the nonlinear function representation of the sound speed.

First, the solution of the problem is approached with a traditional Machine Learning model, a decision-tree algorithm XGBoost. In effect, some important characteristics of the collected data sample are revealed. Moreover, by testing inference capacity of the database with a reliable algorithm, gives an estimate of the properties of the Sound Speed Profile that have the biggest impact on sound propagation. It is proven, that with carefully engineered features, that include a degree of added expert knowledge, a standard model can achieve good accuracy of prediction.

Secondly, a Knowledge Graph is designed to represent the whole context of explicitly stated expert knowledge, using concepts from Hydroacoustics. They are encoded in a form of relational structure connecting actual features of the data into logical categories. In this representation it can be used by the Knowledge Graph Convolutional Network model designed for the problem. A range of tests performed on KGCN proves that using a Graph Neural Network can be feasible to solve the problem, however it also reveals a range of issues regarding model's capability to handle the complexity of problem statement.

Table of Contents

List of Figures

List of Tables

Acknowledgements

During working on this thesis, I have received a lot of support and assistance. First of all, I would like to thank my supervisor dr. ir. J. Sijs, who played an essential role in the process of this thesis work. Without his advice and encouragement, I would not have been able to make this work such a success. Thanks to his trust I had been granted an opportunity to work among highly competent and talented scientists at TNO Acoustics & Sonar, which I would like to acknowledge for their helpfulness and eagerness in answering to my questions whenever needed. I wish to extend my special thanks to Nikol, Fieke and Iris whose help was especially valuable to me at different stages of the project. I would like to thank to J. Fletcher of Grakn for his expertise and assistance in resolving the most challenging problems.

Last but not least, I want to express my gratitude to my family who supported and encouraged me throughout the year, to my friends who expressed interest exceeding my expectations, leading to many inspiring conversations, and to Jazz Café Bebop for being the best place to facilitate all of them.

Delft, University of Technology — Jakub Pietrak
September 10, 2020

Chapter 1

Introduction

Modern developments in the area of data science present a great potential to change the approach in mathematical modelling of the engineering problems. The reason for that is an integration of new tools like *Deep Neural Networks* that are capable of solving increasingly complicated classification and nonlinear regression tasks by approximating the target function directly from the space of collected observations. Unlike most of standard Machine Learning (ML) algorithms that require from a data scientist to tediously shape the feature space, in order to provide the most accurate description of the problem domain [80], Deep Learning (DL) eliminates the need of feature engineering almost completely. It comes at a cost of increased computational power that complex Neural Network (NN) may require. However, with quickly advancing technological progress in semi-conductor industry predicted by yet non-reclining Moore's law, the need for faster, distributed processing units is being continuously satisfied allowing for new breakthrough data-based solutions to appear. Deep Learning was proven to be especially effective in breaking down complex, nonlinear problems on large, multi-variable datasets in variety of formats i.e. time-series in the area of acoustics [51], image processing [55], natural language processing [11], and many more.

This previously unachievable level of abstraction in capturing correlations in the data is possible due to end-to-end training, a greedy optimization search that progresses through multiple interconnected layers of a Neural Networks. With use of, most commonly, gradient-based backpropagation algorithms, it is possible to indicate how a model should change its internal parameters to approximate the target function space. It allows for a model to be optimized as a whole, adjusting the feature map globally, rather then selectively. Deep Neural Networks are thus capable of far-reaching generalisation property that attracts the attention of the Artificial Intelligence (AI) community. Abstract thinking, deductive reasoning and being able to easily and naturally generalise to even very complicated concepts is known to be unique to humans. Many cognitive tests are being made on intelligent species of animals, like chimpanzees or rats, yet most of them lead to the same conclusion: similarities between animal and human abilities are small, dissimilarities large [66]. The concept of gaining one's understanding of the world through non-trivially related observations and abstract connections to the past experiences is closely related to *implicit* or *tacit knowledge*. In contrast to

explicit knowledge, which can be easily expressed by means of writing or verbalising of the reasoning process, the implicit thinking requires placing the information in a wide context. It is a by no means a huge challenge to teach the ways of conceptual thinking, especially to a machine. That is why it is considered to be a the holy grail of computer science a key path forward for modern AI [15].

1-1 Knowledge Engineering in Data Science Applications

As in the real world, explaining a complicated problem may require multiple means to transfer the message (even this report is accompanied by multiple visual aids, references, tables, etc.). In data science and especially in the era of the Internet of Things, approaching a real-life problem will always require dealing with multiple mixed-type inputs, sparsity and non-uniformity of information. Most of the state-of-the-art, high performing NN architectures can be used effectively with dedicated tasks and data formats, but will perform worse on the others. Therefore, Convolutional Neural Networks will be seen in image processing applications, while Recurrent Neural Networks in natural language processing or time-series, because of different mechanisms of transforming the inputs. Stacking up different network architectures to deal with variety of inputs is possible, yet not always effective and it tends to separate, rather than to capture the general *relational structure* of the problem domain. It confronts a scientists with a challenge of developing innovative tools for learning of *heterogeneous knowledge* from different sources of unprocessed heterogeneous data, that are interconnected with each other. Performing an accurate inference of such database, poses a need for a suitable description that would be able to capture the most of encoded information. Traditional formal languages for describing data (i.e. DDL, SQL), that have table-like structure present a limited capability of describing relations between data entities. On the other hand, it is arguably easier to show relational complexity in a form of a *multi-dimensional graph*, where data entities and features are interconnected according to their dependencies. This allows for encoding of *explicit knowledge* information in the database, which can boost the result of automated reasoning. The importance of graph networks has been widely investigated, for their ability to represent complicated structures which contain rich underlying value [13].

1-1-1 Knowledge Graphs

Complexity of the domain is a common characteristics of a real-world problems description. To illustrate this an example of a social network database, that is presented in fig. 1-1. Assume that, having collected some information of James' browsing history, the social network owner tries to categorize this person's interest to personalize the recommendation engine. The data represents very different concepts: visited places and Wikipedia information related to them, James' friends and some events from his life, etc. The data may also come in a variety of formats: continuous numbers and categorical labels, blocks of text, heuristic rules and conditions, distributed GPS locations, time-stamps, images and so on. In fig. 1-1 the available information about James is assembled in an enriched graph representation, where each node is a some data entity or a piece of information, while the arrows show roles that these concepts play in relations that join them. The direction of the arrow has a meaning too - it can represent the influence that one entity has over the other. This set of logical rules

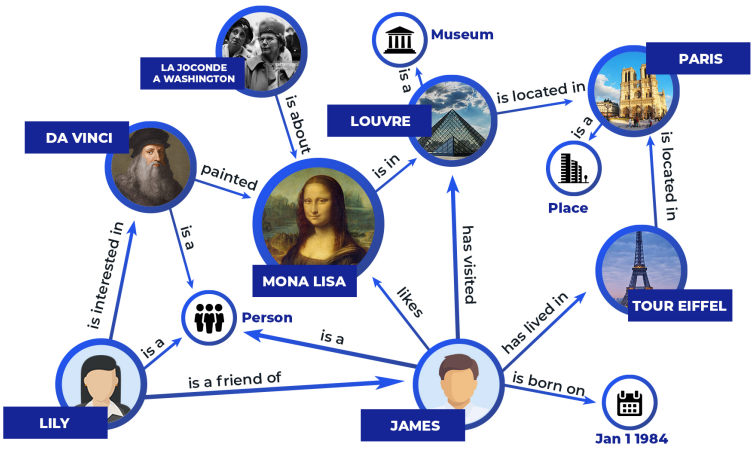


Figure 1-1: Example of a knowledge graph. The graph tries to capture interest detection based on social media connection betweeen Lily and James. Source: [1]

representing relations between data entities is the encoded explicit knowledge. A graph like this might have been retrieved with a specifically designed query from much larger database. If new data is uploaded to the database, the developed schema, using a set of pre-defined rules will automatically label new instances and integrate them into the network. Using the presented example, one can distinguish three main advantages of using a knowledge graph representation for a database, which according to Wilcke et al. [80] are:

- **Encoding knowledge using statements.**
  The statements tie data entities by creating functional relations between them. For example:
  "Lily - *is a friend of* - James", or "Da Vinci *painted* Mona Lisa".

- **Expressing background knowledge in ontologies.**
  Ontology gathers a set of rules governing the creation of links between data entities. In the above example it could be:
  "When: A - *is a* - person; Then: A - *was born on* - date."
  That suggests that there must be a link between a date of birth and a person in any ontology.

- **Reusing knowledge between datasets.**
  The set of rules in ontology, much like the ones described above can be reused with new data. 'A' in this example takes the value of 'James' but that could be as well 'John' and assuming that 'John' has the same characteristics, the rule will still hold.

Graph database representation, recently gains a lot of attention in the deep learning community. The structure has been given a name of a Knowledge Graph (KG) or a GraphNet (GN). It can be seen as a deep learning application of a broader concept in Computer Science called an *ontology*. An ontology is an information object that can be used to make a data model, in terms of its relevant concepts and the relations between them. Ehrlinger and Wöss [29] use the following definition: "An ontology is as a formal, explicit specification of a shared conceptualization that is characterized by high semantic expressiveness required for increased

complexity." The concept of ontologies is adopted in different fields, including the robotics and automation field, as a means to form a semantic model of a knowledge domain [42], that while being easily understandable to a human, it could also be used to train the machines to "think" like us. For the purpose of this paper, terms "knowledge graph (schema)" and "ontology" can be considered to have an equivalent meaning, but to be precise, the knowledge graph is an implementation of the broader ontology theory. Different designs of a Graph Neural Network (GNN)s architecture to perform end-to-end training on the knowledge graphs has been proposed in a few papers during the last decade in [80] discussing its theoretical performance on heterogenous datasets, applied to tasks such as text-based similarity ranking [78], lower-dimensional representation learning [81], clinical decision support [68]. Also there were a few attempts to propose different convolutional architectures [53], reinforcement learning setup [83] and much more, which will be further elaborated on in chapter 4.

1-2 Use Case Introduction: Underwater Sound Propagation Modelling

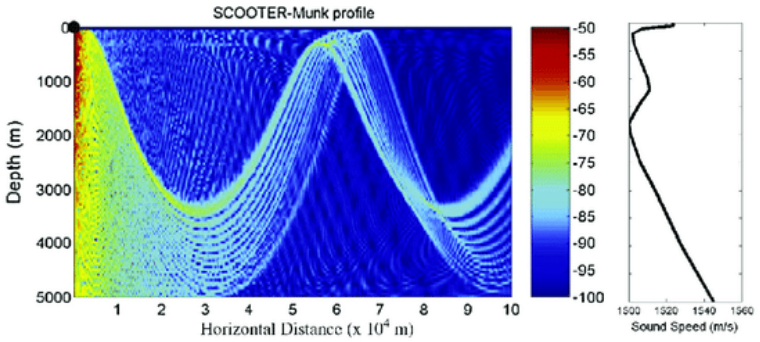


Figure 1-2: Example of an acoustic field calculated with BELLHOP model for a Munk Sound Velocity Profile. Source: BELLHOP User Manual [64]

In this research paper, the engineering application for the knowledge graph model concerns a problem from a domain of acoustic modelling, specifically the underwater sound propagation modelling for a *Sonar System*. The is underwater sonar is used as the primary navigation and detection system for a range of naval and submarine vehicles, including autonomous Autonomous Underwater Vehicle (AUV)s. Design of a reliable sonar is of critical operational importance for these units. For that reason Sonar Performance Model (SPM)s are used for modelling and testing system's properties. In The Netherlands Organisation for Applied Scientific Research (TNO), Acoustic and Sonar department, mathematical models are developed to enable dynamic analysis and control of a sonar system used in a range of underwater vehicles for seabed investigation missions. This study concerns TNO's In-house Developed Ray-based Algorithm (INDRA) model derived from a modified BELLHOP [64] model, that calculates underwater propagation loss of sound based on a Gaussian Beam Tracing (GBT) method. The model takes into account a number of environmental and system design parameters. It is build upon geometric acoustics theory, namely a ray-tracing algorithm. In a simplified case, sound waves used for mapping of a water column can be approximated by *geometric one-dimensional rays*. Assuming that the sonar produces sound waves that are

equally distributed, at a constant amplitude and frequency, the system's performance can be controlled only by the number of rays it produces. Increasing the number of rays is equivalent to increasing resolution of a mapping at the cost of higher power output of an active sonar. In the simulated environment, the achieved accuracy of mapping at growing power levels can be compared by calculating the change in propagation loss at a signal receiver. If this difference is small enough, the power level is considered sufficient for accurate mapping of a water column in a given *sound propagation scenario*. This then is considered to be the optimum number of rays needed for this underwater environment.

1-3 Problem Statement: Sound Modelling with a Graph Neural Network

Taking all the above aspects into consideration, an outline of the method unifying graph theory with ocean acoustic may now be proposed. The thesis concludes a research project for development of a Graph Neural Network, that could operate on Sonar Performance Model data and perhaps in the future in real life sonar applications. For the time being, the GNN should be able to predict the optimal simulation output - minimum required number of rays - based on the input features provided to BELLHOP. The ground truth for prediction is the actual BELLHOP output. It states an additional requirement of designing a functional ontology to express the parameters of the considered Sonar Performance Model. Therefore, the end-goal of the proposed research can be expressed in a following research question:

RESEARCH QUESTION:

Design of a Graph Neural Network for the Sonar Performance Model to predict the optimal sonar resolution, characterized by a stable sound propagation calculated in simulated ocean environment with BELLHOP ray-tracing algorithm.

1-3-1 Thesis Outline

To handle the problem effectively it has been decided that prior to development of a Graph Neural Network, which is a complex and multi-stage problem in a largely untested and innovative deep learning domain, first it is necessary to attempt predicting the target variable with a more reliable ML algorithm. This should allow for building up on the understanding of the problem going beyond the statistical analysis of collected simulation data. It also provides a *baseline score* that can be used to measure the effectiveness of the GNN method. For that reason, in the initial stage of the research, beside statistical data analysis, a Machine Learning decision-tree based model was developed with use of XGBoost [76] algorithm. This model is used to perform the inference of the BELLHOP database and, in fact, to prove that the problem can be solved effectively with a data-based machine learning approach.

Before the results of this approach are presented in chapter 3, in chapter 2 the explanation of the main concepts of Acoustic Underwater Sound Propagation is provided. Within this chapter the equations governing geometric ray-tracing algorithm are explained. Also, it discussed

the significance of different input features of the BELLHOP model and the simulation setup in which the database for the problem was obtained.

Then, the ML-dedicated chapter 3 focuses on the process of designing XGBoost model. The chapter begins with statistical analysis of the collected database. It is quickly revealed that the input provided to BELLHOP can be problematic to feed into a Machine Learning algorithm due to mixed format of the input features. The problem indeed concerns learning from heterogenous data sources. Also, the dataset is characterized by a heavily unbalanced distribution. The chapter discusses different approaches to feature vector representation, design, training and evaluation of the decision tree model that can lead to obtaining a satisfactory and reliable prediction score.

Continuing, chapter 4 describes the core problem of the research, which is the development of the SPM ontology and the Graph Neural Network to work with it. The theory for deep learning framework is laid by exploring deep learning solution for graphs and distributed networks, where the core concepts are rooted in the *relational inductive bias* proposed by Battaglia et al. in GraphNet paper [15]. This theory has been adapted by the team of Grakn [75], whose distributed graph database processing engine is used as the main tool in the development of the ontology and later also in Knowledge Graph Convolutional Network (KGCN) model design. The contents of the chapter gives a comprehensive introduction to graph-based deep learning theory and network architecture designs, together with the functional description of Grakn-specific language (Graql) and the multi-element Python code library that needs to be integrated in the model. The results of extensive testing of the model can be found in the final section of the chapter. Finally, a comprehensive summary of the results is provided in chapter 5, together with the reflection on the method and proposed direction for the further research of the problem.

Chapter 2

Underwater Sound Propagation Theory

Due to characteristics of the environment of underwater missions, sonar systems are necessary choice for the primary sensing system of the vehicle, responsible for navigation and environment monitoring. The lack of light and possible turbidity underwater makes using electro-optic sensors (e.g., cameras) challenging. Furthermore, electromagnetic waves are heavily damped in water [74], making it impossible to use radar. Acoustic waves, however, can carry farther than thousand kilometres [58]. Therefore, often acoustic sensors such as sonar are used for perception underwater. Quite often playing the role of a critical subsystem for operational readiness, the importance of having a precise Sonar Performance Model (SPM) is self-evident. Required system parameters are obtained, as it is usually achieved in the engineering field, through extensive methods of system simulation, testing, validation and consecutive design iterations. The need to optimize of a sonar model is well captured in a quote from "Underwater Acoustic and Modelling" by Paul Etter [31]: "The ultimate purpose of sonar performance modelling is two-fold. First, advanced sonar concepts can be optimally designed to exploit the ocean environment of interest. Second, existing sonars can be optimized for operation in any given ocean environment."

The content of this chapter aims to outline the most critical elements of an acoustic model that have an influence on underwater sound propagation scenarios investigated in this research. Section 2-1 contains a brief description of commonly used types of acoustic models, followed by a breakdown of variables used to describe the phenomenon of sound wave propagation in the ocean environment. Sections 2-1-1 to 2-1-3 take a closer look at the speed of sound in the water. There is a range of numerical and statistical methods developed over the past decades with a goal to synthesize a generalised environmental ocean model similar to i.e. atmosphere models used in Aerospace industry or weather forecasting. Section 2-2 provides a mathematical derivation of geometric ray tracing and its extension to the beam tracing methods used in the provided acoustic model BELLHOP [64] that was used to generate the database of sound propagation scenarios. This database is used as an input for the data-based models developed later. Finally, in section 2-3 specifications of the simulation environment are

described, highlighting the most important features such as: input and output parameters, convergence criterion and explanation of the algorithm used to generate the dataset for data-based solutions developed in the later process.

2-1 Elements of a Hydroacoustic Model

There are three types of underwater acoustic models: environmental models, basic acoustic models and sonar performance models [31]. The relationship between these models is illustrated in fig. 2-1 and reveals gradual build-up with respect to phenomena taken into account. The more complex representation, the more applications become more system specific (i.e. as one progresses from environmental models toward sonar performance models). BELLHOP, is a type of an SPM investigated in this study, for simplicity to be addressed as TNO's SPM or simply the SPM, due to its dedication to be used with specific TNO's systems.

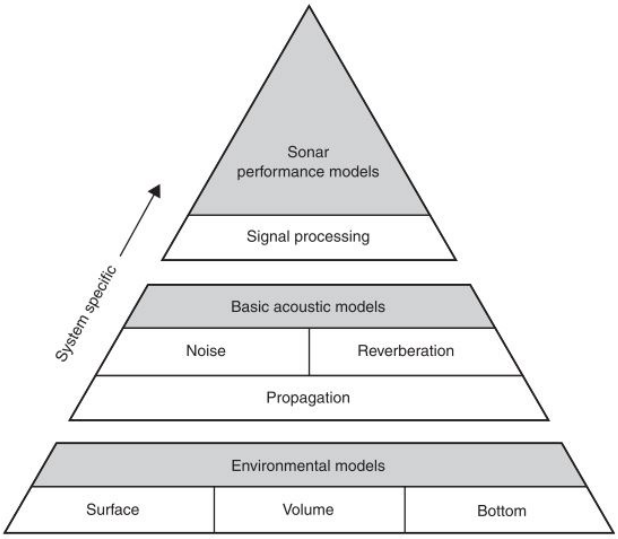


Figure 2-1: Generalized representation of relationship between environmental models, basic acoustic models and sonar performance models. Source: [31]

A brief explanation of different model types, based on the categorization of Etter [31] is given below:

- **Environmental models:** Empirical models approximating conditions of the ocean environment, e.g., sound speed, absorption coefficients, surface and bottom reflection losses and backscattering strengths. They may take into account factors such as wind speed and wave height at sea surface, temperature distribution and salinity of water volume, composition and roughness of the sea bottom.
- **Basic acoustic models:** With a discrete description of the environment, the estimate of acoustic propagation can be obtained. The noise model quantifies the behaviour of noise sources between sonar and receiver. Modelled reverberation quantifies the scattering effect of a sonar pulse.

- **Sonar performance models:** Combines the information of environmental and basic acoustic models, together with system-specific characteristics embedded in the higher-level models to assess utility of a particular sonar system.

2-1-1 Speed of Sound in the Ocean Environment

The estimate of speed with which sound waves travel in a given underwater environment is an essential parameter for any acoustic model. In the most basic form speed of sound, usually denoted by c is a function of isothermal compressibility factor K, ratio of specific heats of sea water at constant pressure and constant volume γ, and the density of sea water ρ:

$$c = \sqrt{\frac{\gamma}{K\rho}} \tag{2-1}$$

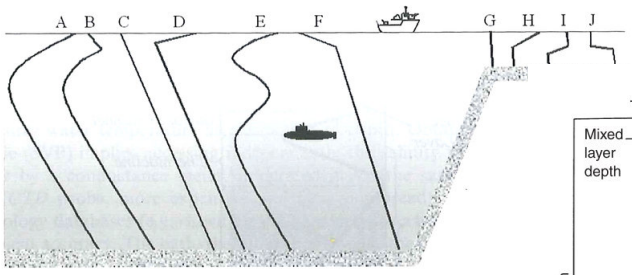
Although the solution to Equation (2-1) looks rather straight-forward, in practice it is very difficult to estimate its true value due to appearance of non-measurable constants in the equation. On the deeper level, presence of salt in sea water makes the water-salt solution a binary fluid, a mixture which microstructure and flow are strongly coupled [19]. Salinity level has a major effect on a number of oceanic thermodynamic parameters including compressibility, refractive index, thermal expansion, freezing point and temperature of maximum density [31]. Water salinity can be measured or estimated based on temperature-salinity diagrams of a given water mass. There were many attempts to express speed of sound in relation to easier to obtain parameters, leading to a range of empirical relations with separate "domains of applicability", i.e. one of the "shorter" ones the 9-term Mackenzie (1981) formulation expresses the speed of sound as a function of temperature T in degrees Celsius, salinity s in parts per thousand and z which is depth in meters:

$$\begin{aligned} c(T,s,z) =& 1448.96 + 4.591T - 0.591T - 0.05304T^2 + 2.374 \times 10^{-4}T^3 + 1.34(s-35) \\ &+ 0.0163z + 1.675 \times 10^{-7}z^2 - 0.01025T(s-35) - 7.319 \times 10^{-13}Tz^3 \end{aligned} \tag{2-2}$$

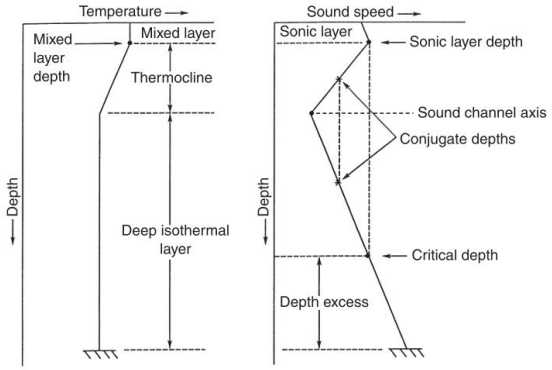
Effectively there is no one coherent model that would enable calculating speed of sound propagation in sea water under any arbitrary conditions and often the most accurate estimate can be obtained by using expensive measurement tools *in situ*. Two most common ways of obtaining sounds velocity measurements are depth-velocimeters and bathythermographs, or eXpendable BathyThermograph (XBT) probe [58].

2-1-2 Sound Speed Profile

For some applications, (i.e Anti-submarine Warfare) such accurate measurements may be required, but often the environment can be approximated as horizontally stratified; sound velocity only depends on depth, which leads to range-independent propagation assumption and greatly simplifies the modelling. Such profile acts as a waveguide for the sound wave propagating underwater. This approach is also adapted in TNO's SPM model, which makes use of LEVITUS database. LEVITUS [6] is an extensive collection of major ocean parameters at the annual, seasonal, and monthly time scales. To give a better insight in this important model feature, the mechanism of obtaining some common Sound Speed Profiles will be explained. Sound Speed Profile (SSP) may be broken apart to a several characteristic parts [58]:



(a) Overview of different generic sound velocity profiles. Depth is presented on the y-axis and the velocity variations in the x direction.[58]



(b) Schematic relationship between temperature, depth and sound-speed profiles in the deep ocean. [31]

Figure 2-2: Representations of the speed of sound as a function of depth - Sound Speed Profile

- A homogeneous layer (*mixed layer*) of constant sounds velocity, often present in the first few meters. It corresponds to the mixing of shallow water through surface agitation.

- A *surface channel*, corresponding to a sound velocity increasing from the surface down. This channel is often due to a shallow isothermal layer appearing in winter conditions, bit can also be created by water that is very cold at its surface (e.g., ice melting), or by an input of fresh water near river estuaries.

- A *thermocline*, monotonous variation of temperature with depth. It is most often negative (temperature deceasing usually from the surface to the bottom), and then induces a velocity decrease with depth. It can be seasonal (close to the surface) or permanent.

- A *deep channel*, a sound-speed minimum between the negative sound speed gradient of the thermocline and the positive sound speed gradient of the deep isothermal layer. The depth corresponding to this sound-speed minimum is referred to as the sound channel axis. The average depth-velocity profile of a large ocean basins shows a deep channel between a few hundred meters and 2000 meters. At high latitudes, the deep isothermal layer extends nearly to the sea surface. That is, the sound channel axis shoals as one approaches the polar regions [31].

- A *deep isothermal layer*, at constant temperature. Sound velocity increases linearly with depth, because of hydrostatic pressure. The deper layers in the ocean are approximately isothermal. So are the sound velocity profiles in closed seas (e.g., the Mediterranean) or in shallow waters in winter conditions.

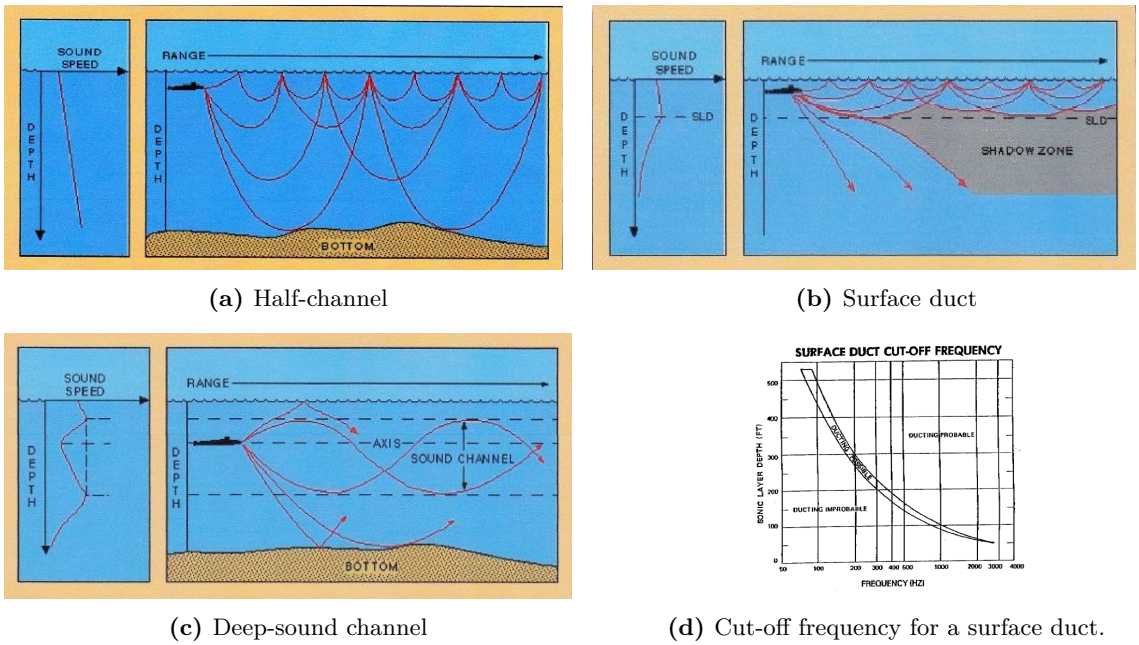
2-1-3 Special Propagation Modes

There are a few distinct propagation modes that can be characterized by low achievable propagation loss (and thus high range) through boundary layers and sound-speed profile interactions. Those are called *ducted propagation modes* fig. 2-3. They are especially interesting, because of possibility of using them for carrying sound waves at large distance without loss of energy. Ducted modes may emerge on SSP that has some special characteristics regarding the magnitude and the position of consecutive extrema of the sound speed function. In profiles where a global minimum does not lay at one of the boundaries - there is a distinct convex part - a sound channel axis can be identified. It has a corresponding critical depth that is a depth below critical axis at which the sound speed equals the near-surface maximum value. The near-surface maximum value of sound speed is usually located at the first local maximum below the surface called Sonic Layer Depth (SLD). The vertical distance between the critical depth and the sea floor is referred to as the depth excess. Other pairs of points can be identified on the sound-speed profile that have the same value of sound speed but which lie on opposite sides of the sound channel axis. Such pairs are referred to as conjugate depths. So called full Deep-Sound Channel (DSC) or SOund Fixing and Ranging (SOFAR) channel is formed around the sound channel axis and, for the purpose of this discourse [1], will be limited above the axis by the SLD and below the axis by the sea floor, as in i.e fig. 2-2b, where the critical depth is a conjugate of the SLD. Under additional condition cited in i.a. [8], surface ducts and deep sound channels are trapping the sound waves above a "cut-off frequency", that depends on the channel width and the frequency of the wave. This is related to the energy of the propagated sound wave and the amount of refraction happening at the channel layer boundary. For a source with a radiation frequency of 1000Hz, like the one considered here, the minimum width of a surface duct is equal to 100ft or 30.48m, which can be measured from fig. 2-3d. With such a high frequency wave, the deep channels can be considered as always trapping. A representative case of SSP with distinct DSC and SLD can be seen in fig. 2-2a B, while DSC alone is depicted in A. It may also happen that in simple cases SSP can be represented by linear fig. 2-2a C or a piecewise-linear fig. 2-2a D function especially in the shallow water fig. 2-2a G-J, due to limited number of measurement points. The direction and propagation is then simply dictated by the strength and sign of the gradients of consecutive profile segments.

Whether sound propagation behaviour is captured within one of these special modes is highly dependent on the depth of source and receiver, their relative position with respect to each other and to the segment sound-speed profile. For the traverse propagation through the waterway the three important modes are: surface duct, propagation in the deep-sound channel and half-channels.

Half-channel First special case to consider is when the SSP is simply a straight line spanning from the water surface to the bottom layer, as presented in fig. 2-3a. This propagation mode is called a half-channel and it provides a relatively good propagation range due to minimized bottom interaction, especially when the water surface is flat and highly refractive. The

[1] For complex sound speed profiles with multiple distinct minima and maxima, it is possible to characterize multiple sound channels. Identification of these secondary ducts is out of scope of this research. Global minimum deep-sound channels are the most significant influence to the overall sound propagation.



(a) Half-channel

(b) Surface duct

(c) Deep-sound channel

(d) Cut-off frequency for a surface duct.

Figure 2-3: Ducted propagation modes. Source [8].

opposite case may as well take place if the SSP gradient is negative and directs all sound waves towards the bottom, which leads to high absorption and limited propagation.

Surface duct Secondly, fig. 2-3b presents the surface duct. It is a zone bounded above by the sea surface and below by the SLD. Within the surface duct, sound rays are alternatively refracted on the SLD and reflected from the surface. A surface duct exists when the negative temperature gradient within it does not exceed a value determined by the effect of pressure on sound speed. Specifically, the surface duct is characterized by a positive sound-speed gradient [31]. In the case when source is placed above SLD, surface duct is trapping large portion of emmitted sound waves, however if the source is placed below it, it may refract the rays towards the bottom leading to higher propagation loss. A special case happens when source is placed directly at SLD, leading to rays being refracted in opposite directions away from the SLD axis. This leads to the case of highly adverse *dipolar spreading*, where sound intensity is lost at the rate of $1/r^4$ instead of cylindrical rate of $1/r^2$. [4]

Deep-sound channel Another significant mode of propagation to be investigated is the deep-sound channel fig. 2-3c. This internal sound channel allows for sound transmission entirely via refracted paths, which means that a portion of the acoustic power radiated by the source in the channel propagates to long ranges without encountering reflection losses at the sea surface and the sea floor. The potential of deep sound channels for long-range propagation has been proven in experimental setup. Because of the low transmission loss, acoustic signals from small explosive charges in the deep sound channel have been recorded over distance of thousand kilometres, in some case even half way around the world [10]. Depending upon the ocean environment, propagation over combinations of paths may be possible for any given source-receiver geometry.

There are numerous other factors that may locally change the composition of isothermal layers, including water currents and thermal fronts [58]. Also water basins located at high altitudes, where there is cold water flowing from melting ice, or in places of large mass transfers - near river estuaries and straits between large ocean basins (i.e. Gibraltar) can present significantly different SSP due to mixing of the water layers. Local sound speed models are an object of observations, acoustic probe measurements and statistical studies, that when needed, can be found for a specific water basin [28, 12, 69]. Identification of the above mentioned propagation modes will play an important role in classification of simulated scenarios, as they provide a strong guideline for the mode in which sound travels through the water.

2-2 Introduction to the Hydroacoustic Mathematical Model

Having introduced the components of the SPM, this chapter is focused on more in-depth mathematical description of the algorithm used in the model to estimate acoustic wave propagation loss in the simulated ocean environment. Understanding of the mathematical derivation of the ray-tracing theory is not strictly required to follow any of the data-based solutions derived in chapters 3 and 4, however it may make it easier to interpret the output of the simulation model. For the reader interested only in the description of BELLHOP configuration and preliminary analysis of the collected data samples, it is advised to move directly to section 2-3.

2-2-1 The Spherical Wave Equation

In mathematics and physics a core of acoustic wave propagation is the linear partial differential equation named after Hermann von Helmholtz:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right)$$
$$\left(\nabla^2 - \frac{1}{c^2} \frac{\partial^2}{\partial t^2} \right) u(\mathbf{r}, t) = 0 \tag{2-3}$$

where $u(\mathbf{r}, t)$ is the pressure of a wave propagating with time t in the three-dimensional space $\mathbf{r} = \langle x, y, z \rangle$. $\nabla = \left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} + \frac{\partial u}{\partial z} \right)$ denotes the Laplacian operator and c is the local sound propagation velocity. Helmholtz equation can be solved by assuming that the wave equation is in fact separable, such that $u(\mathbf{r}, t) = A(\mathbf{r})T(t)$. Furthermore, restricting solutions to waves that oscillate in time with well-defined constant angular frequency ω in all three dimensions (isotropic propagation), one can arrive at the scalar solution of Helmholtz equation:

$$u(\mathbf{r}, t) = \frac{A}{r} e^i(\omega t + kr) \tag{2-4}$$

The wave fronts are spherical and centred around the origin r_0, here k denotes the wave number ω/c. From this one can observe that the peak intensity of spherical wave oscillation,

defined as the square of its amplitude A, follows the inverse-square law and decreases at the rate $1/R^2$:

$$I = |u(\mathbf{r}, t)|^2 = \frac{|A|^2}{r^2} \qquad (2\text{-}5)$$
$$I \propto 1/r^2$$

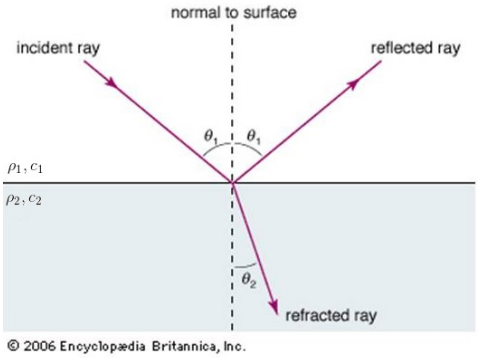
The intensity of the wave is the magnitude of interest in majority of numerical models assessing source-receiver performance based on their Signal-to-Noise Ratio (SNR). Computational ocean acoustics rarely relies on solving the wave equation, as it requires resource-intensive calculations in discretized space (finite element methods). However, a suitable alternative is based on *geometrical acoustics* in which sound is supposed to act as rays and the wavelength of the sound is neglected. This means that all the wave-based phenomena, such as diffraction and interference, are missing in those methods. This leads to some artefacts, i.e. perfect shadows and caustics, that yet can be tackled with use of extended ray theories or beam tracing [65] (which will be elaborated on in Section 2-2-3). These methods are called ray-based since they often use some kind of rays or particles that are reflected at the surfaces. One major difference to wave-based methods is that they typically compute only sound energies instead of sound pressure or particle velocity used in the wave-based methods [72]. This leads essentially to a high-frequency approximation, they are only valid when the scale of variation in the medium is larger then the wavelength considered. The method is sufficiently accurate for applications involving echo sounders, sonar, and communications systems. However, it has also been proven that ray-method can be used at low frequencies [48]. Some of the most common approaches are based on ray theory, modal expansion and wave number integration techniques.

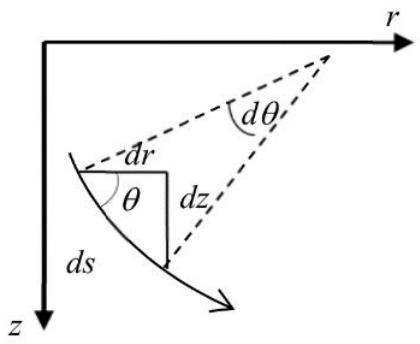
2-2-2 Ray Tracing Method

Ray acoustics and ray tracing techniques are the most intuitive and often the simplest means for modelling sound propagation in the sea [47]. This section aims to provide the derivation of *geometric transmission loss* for the standard ray tracing algorithm, to prove that - under a range of assumptions - the path of the sound wave and its intensity can be approximated knowing local changes of depth-dependent sound speed (gradient) $g(z) = dc(z)/dz$ and the distance from the source. In brief, geometrical acoustic aims at modelling the structure of an acoustic field as a set of rays falling under three basic principles [58]:

1. refraction of the propagation direction by velocity changes, according to the Law of Snell-Descartes
2. specular reflection, also known as regular, mirror-like reflection, at the interfaces
3. intensity losses along the rays, through geometric divergence, which consists of spherical divergence being modified by refraction, through absorption along the paths and through reflections at the interfaces

Derivation of the equations governing these three acoustic phenomena are provided in this section. Finally, the ray-tracing approach extends, however indirectly, to the Gaussian Beam





(a) Snell's law: reflection and refraction of a plane wave, due to change in sound velocity c at the inter- face. **(b)** A small segment of a ray path in a isotropic medium with arc length ds

method briefly discussed in the last section. Underlying idea behind all ray tracing methods is that a plane wave, approximated by a one-dimensional line - a ray - undergoes a refraction and reflection in the medium, at the interface of varying sound speeds $c(z)$ according to the simple Snell's law of refraction. The interface can be understood as a boundary between two infinitesimal segments of an SSP profile. That also indicates implicit dependence on depth z (see section 2-1-2).

$$\xi = \frac{\cos\,\theta_1(z)}{c_1(z)} = \frac{\cos\,\theta_0(z)}{c_0(z)} \tag{2-6}$$

ξ is a *ray parameter* and it is constant only if the interface is flat. The ray approximation enforces also that the ray undergoes specular reflection (also known as regular, mirror-like reflection) at the interfaces [58]. If a ray with horizontal angle θ_{in} strikes an interface with inclination α, the reflected ray is changed to $\theta_{out} = \theta_{in} + 2\alpha$ and the ray parameter is no longer constant then. Instead it can be calculated as:

$$\begin{aligned} \xi_{out} = \frac{cos(\theta_{out})}{c} = \frac{\cos(\theta_{\theta_{in}+2\alpha})}{c} = \\ = \xi_{in}\cos(2\alpha) \pm \frac{\sqrt{1-\xi_{in}^2 c^2}}{c}\sin(2\alpha) \end{aligned} \tag{2-7}$$

See that c remains constant, because the ray is reflected back in the medium with the same parameters. It can be seen that continuous changes of the speed of sound along depth z, will lead to progressively changing initial direction of the wave. The orientation depends only on local sound velocity. It will cause the rays to travel along curved paths, as illustrated in fig. 2-5. From the same illustration the radius of curvature R, defined as the ratio between an increment in the arc length and an increment in the angle, by a trigonometric relation can be expressed in terms of dz:

$$R = \frac{ds}{d\theta} = \frac{1}{\sin\theta}\frac{dz}{d\theta} \tag{2-8}$$

Further knowing that when the sound speed varies with depth, the ray angle $\theta(z)$ is also a function of depth, according to Snell's law, the equation can be rearranged to:

$$R(z) = -\frac{c(z)}{\cos\,\theta(z)}\frac{1}{g(z)} = -\frac{1}{\xi g(z)} \tag{2-9}$$

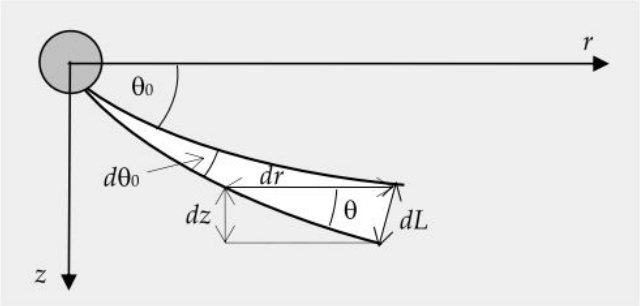


Figure 2-5: The principle of intensity calculations: energy radiated in a narrow tube remains inside the tube; r_0 represents a reference distance and θ_0 is the initial ray angle at the source; $d\theta_0$ is the initial angular separation between two rays; dr is the incremental range increase; is the angle at the field point; dz is the depth differential; and dL is the width of the ray tube. [47]

Which in effect proves that the radius of curvature of the ray path can be estimated with respect to refraction ray parameter ξ and local changes in the sound speed gradient $g(z)$.

To calculate losses from geometric spreading according to ray theory, consider the plane wave that undergoes a circular refraction (due to velocity change in the medium). Remind the fact that intensity of radiant energy is the power transferred per unit area. In the basic form Transmission Loss (TL) is defined as 10 times the log (base 10) of the ratio of the reference intensity I_0, measured at a point 1m from the source, to the intensity I, measured at a distant point, and is expressed in units of decibels (dB):

$$TL = 10 \log_{10} \frac{I_0}{I} \tag{2-10}$$

Due to cylindrical symmetry of this refraction the acoustic intensity of a ray can be calculated using the principle that power within the ray tube remains constant within its boundaries. This is illustrated in Figure 2-5 showing two rays that mark the boundaries of a segment of a ray tube with the separation angle $d\theta_0$. At a reference distance r_0 from the source, the intensity is I_0. Taking into consideration the cylindrical symmetry about the z axis, the power ΔP_0 within the narrow angle $d\theta_0$ is [47]:

$$\Delta P_0 = I_0 2\pi r_0^2 \cos\theta_0 d\theta_0 \tag{2-11}$$

Analogically at horizontal distance $r > r_0$, the intensity is $I < I_0$ and the displacement of the ray tangent to the tube circumference is dL , there the power is:

$$\Delta P = I 2\pi r dL \tag{2-12}$$

Since the power in the ray tube does not change, the above equations are equal, hence when solved for the ratio of intensities they result in relation:

$$\frac{I_0}{I} = \frac{r_0^2}{r} \cos\theta_0 \left| \frac{d\theta_0}{dL} \right| \tag{2-13}$$

Instead of using tangential displacement dL in eq. (2-13), it is more convenient to replace it by trigonometric identity in terms of vertical ray displacement dz:

$$dz = \left| \frac{dL}{\cos\theta} \right| \tag{2-14}$$

resulting in:

$$\frac{I_0}{I} = \frac{r_0^2}{r} \frac{\cos\theta_0}{\cos\theta} \left| \frac{d\theta_0}{dz} \right| = \frac{r_0^2}{r} \frac{c_0}{c} \left| \frac{d\theta_0}{dz} \right| \tag{2-15}$$

The last expression in eq. (2-15) is obtained by again using Snell's law. The absolute values are introduced to avoid problems with regard to the signs of the derivatives. By inserting eq. (2-15) into eq. (2-10), the geometric transmission loss becomes:

$$TL = 10\log_{10}\left(\frac{r_0^2}{r}\right) + 10\log_{10}\left|\frac{d\theta_0}{dz}\right| + 10\log_{10}\left(\frac{c_0}{c}\right) \tag{2-16}$$

Note that in this treatment, the transmission loss includes the geometric spreading loss, and the absorption loss must be included separately. The geometric transmission loss consists of two parts. The first term represents the horizontal spreading of the ray tube and results in a cylindrical spreading loss. The second and third terms represent the vertical spreading of the ray tube caused by the depth gradient of the sound speed [48]. In practice, ray tracing model divides the water column into finite number of slices along depth Δz where the SSP can be approximated as linear such that the sound speed in z_i becomes $c(z) = c_i + g_i(z - z_i)$. Subsequently ray paths are decomposed into circular segments for which radius of curvature and travel time can be calculated by numerical integration of equations:

$$r_2 - r_1 = \int_{z_1}^{z_2} \frac{dz}{tan\theta(z)} = \int_{z_1}^{z_2} \frac{\cos\theta(z)dz}{\sqrt{1-\cos^2\theta(z)}} = \int_{z_1}^{z_2} \frac{\xi c(z)dz}{\sqrt{1-\xi^2c^2(z)}} \tag{2-17}$$

$$\tau_2 - \tau_1 = \int_{z_1}^{z_2} \frac{dz}{c(z)\sin\theta(z)} = \int_{z_1}^{z_2} \frac{dz}{c(z)\sqrt{1-\xi^2c^2(z)}} \tag{2-18}$$

2-2-3 Method Extension: Gaussian Beam Theory

Extension of ray tracing theory to Gaussian Beam method aims to overcome major drawbacks of geometric ray approximation, that is lack of wave-specific phenomena such as diffraction. Looking at eq. (2-16) one can realize that in cases when $\theta = 0$ or $dr/d\theta_0 = 0$, the equation outputs infinite intensity. The first condition signifies a turning point where the ray path becomes horizontal, while second occurs at points where an infinitesimal increase in the initial angle of the ray produces no change in the horizontal range traversed by the ray [46]. The locus of all such points in space is called a caustic. Due to limitations of geometric transmission loss standard ray tracing method produces certain artefacts, e.g., perfect (abrupt) shadow zones and infinite energy caustics. To overcome that issue, BELLHOP model designed by Michael Porter and Bucker [64] (based on the seismological work of Cerveny. et al. [30]) implements Gaussian and hat-shaped beams, with both geometric and physics-based spreading laws. The construction begins with integration of standard ray equations to obtain the central ray of a beam. Beams are then constructed about the center-rays by integrating a pair of auxiliary ordinary differential equations that govern the evolution of the beam in terms of beam width

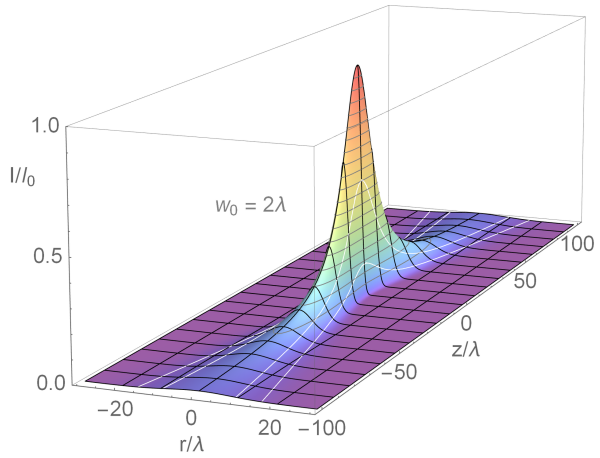


Figure 2-6: Gaussian beam profile. Source: Wikipedia

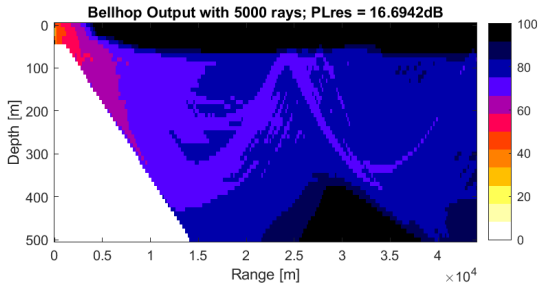
and curvature as a function of arc length. In effect sound wave representation is defined by a Gaussian distribution, or other symmetric function, as presented on fig. 2-6. The resulting pressure field (rather than line) describes a beam. Additionally, the distribution varies in a Gaussian fashion as a function of normal distance from the center-ray of the beam which allows for smooth transitions more alike to the real physical wave behaviour. The derivation of auxiliary equations falls outside of scope of this paper, but it can found in the appendix of Porter(1987) [65]. Due to the fact that geometric ray representation has now been replaced by a statistically varying curve with Gaussian statistics [32], also the TL calculation of eq. (2-16) undergoes a change. A stochastic term representing variance of intensity σ_I is added to the basic TL equation, which results in BELLHOP Transmission Loss:

$$TL_{uncertain} = 10 \log_{10}(I + \sigma_I) \tag{2-19}$$

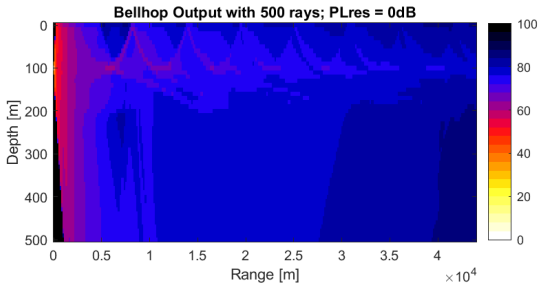
The influence of this term depends on parameters related to the initial beam shape and beam evolution and the sum of intensities of each ray weighted by the Gaussian factor. [59]

2-3 Description of BELLHOP Acoustic Model

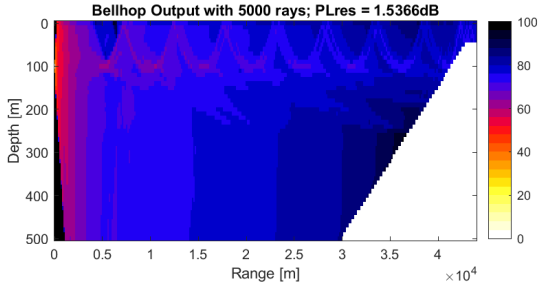
This section subjects to a complete description of the acoustic model BELLHOP utilized for generating a database of underwater sound propagation scenarios. The same exact database provides an input for the data-driven models developed in the following chapters. In a very short description, the whole process can be summarized, to give a better overview of the research direction to the reader. In BELLHOP, each propagation scenario is denoted by a unique set of parameters defining characteristics of simulated ocean environment, such as: depth-wise location of the sonar, shape and type of the bottom boundary or an SSP function, just to mention a few (the full list is provided in table 2-1). With use of beam-tracing algorithm explained above, the algorithm simulates the sound waves emitted from the sonar source, precisely their path and the corresponding Transmission Loss. The calculation is repeated with increasing sonar resolutions, represented here by the number of geometric rays emitted from the source. By comparing the result with a very high resolution run for the same scenario and under certain conditions that will be explained below, it is able to estimate



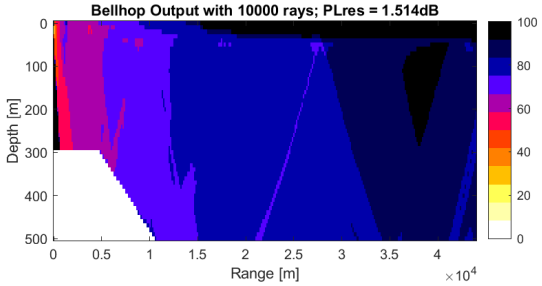
(a) A down-slope scenario with a visible deep sound channel propagation.



(b) A flat bottom scenario with a visible surface duct.



(c) An up-slope scenario with rays captured within the surface duct.



(d) A shallow negative gradient SSP causing the rays to bounce between the bottom and the sonic layer.

Figure 2-7: A few examples of captured propagation modes in BELLHOP simulated environment. The source is placed on the left side of the water column and an array of receiver measures the intensity on the opposite end.

if, with a current resolution, mapping of the acoustic field is accurate enough. If so it is a minimum number of rays to map this simulated environment. The data-driven solutions developed for the presented use case, focus on the reconstruction of minimal resolution found in BELLHOP, by performing inference of the feature space aka input parameters used in the simulation environment.

A disclaimer has to be made, that the investigated model is parallel to a standard BELLHOP of Porter and Bucker [64] but includes a range of modification introduced by TNO experts to improve the stability of calculations and convergence of solutions. Unfortunately details of these changes cannot be disclosed, even so, without loss of generality all the derivations provided in the previous sections remain applicable. The example output of the algorithm presented in section 2-3 shows a few distinct propagation modes discussed in theory in the sections above.

2-3-1 BELLHOP Simulation Parameters

A complete list of inputs that are specified in the BELLHOP simulation setup are listen in table 2-1. System-specific parameters considered in the TNO's SPM include a sonar properties such as: bandwidth, ray cone angle, source-receiver geometry, and most importantly - the number of emitted rays. That is a measure of the resolution of the model and the target variable that will be predicted by data-driven models later. It is important to explicitly define problem setup to avoid confusion.

Table 2-1: BELLHOP simulation inputs. 13 parameters in total.

Var.	Range of Discrete Values	Unit	Description
n_{ray}	500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 6000, 7000, 8000, 9000, 10000, 12500, 15000, 20000	-	number of emitted rays
d_{src}	15, 30, 50, 70, 100, 150, 200, 300	m	source depth
f	1000 (const.)	Hz	sonar wave frequency
θ	20 (const.)	deg	half-beam aperture
d_{min}	50, 150, 250, 350, 450, 600, 750, 900, 1050, 1200	m	minimum depth of the water column
d_{max}	50, 150, 250, 350, 450, 600, 750, 900, 1050, 1200, 1500	m	minimum depth of the water column
s	-2, 0, 2	deg/m	slope inclination
r	44000 (const.)	m	length of the water column
Δr	146.67 (const.)	m	water column slice in the length-wise direction (fixed grid size 300)
Δz	10 (const.)	m	water column slice in the depth-wise direction (grid increases with d_{max})
C_r	1.18, 0.99	-	bottom refraction coefficient
C_a	0.99, 0.90	-	bottom absorption coefficient
SSP	24 unique Sound Speed Profile vectors	-	SSP function taken from LEVITUS [6] 6 randomly picked locations, 4 meteorological seasons

- In passive sonar: the target is the sound source, and the sonar is the receiver.
- In active sonar: there is a sonar source, a target, and a receiver.

In BELLHOP simulation model the sonar is in the active configuration, with a column of receivers located at the end of a 44km long waterway. Receivers are placed at a grid extending from water surface to bottom of the sea, spaced at every 50m from each other. The length of the waterway and hence displacement between sonar and receivers does not change between scenarios. There is no specific target, or obstacle between source and receiver. Thus in the current use case, the direct path propagation loss between a source and receiver is the modelled value of interest. Source may be located at different positions along depth in the beginning of the waterway. It is worth to mention that due to additional constraints on scenario generating algorithm in the provided model, the bathymetry is created in a symmetric manner, which is illustrated on fig. 2-8. The shape of the bottom boundary layer, marked by coloured lines in the plot, is centred around mid-point range of 22000m, thus positive and negative slope scenarios are perfectly symmetric for the same min/max depth values. Flat bottom with slope $s = 0$ scenarios are created only until the maximum depth of 1200m. The sea surface

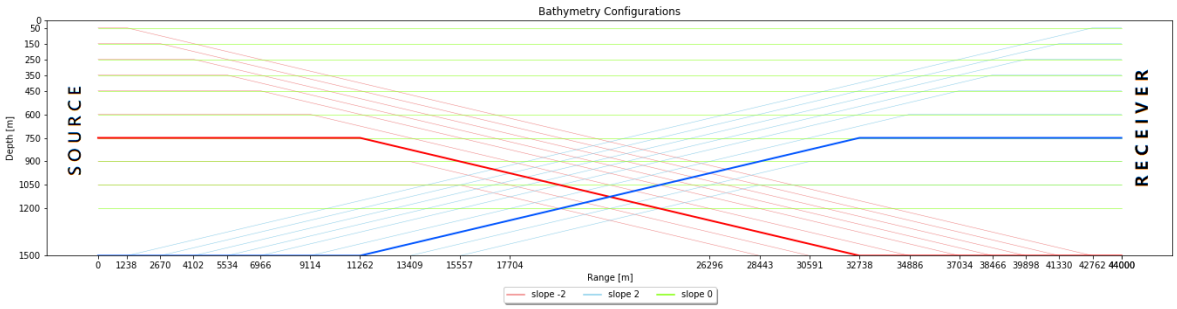


Figure 2-8: BELLHOP bathymetry configurations

is considered to be a flat and fully reflective boundary. The bottom boundary refraction and

absorption are taken into account by introducing a variable set of coefficients corresponding to sandy (reflective) or muddy (absorptive) bottom types. Finally, an important input feature is an SSP vector obtained from the provided database, as a column vector of 24 sound speed values placed on an unequally spaced grid between 0-1500m (grid is dense close to the surface and becomes more sparse towards the bottom). Most importantly, it is the only non-scalar parameter in the input. This will have major consequences for building this feature representation in the input vector for the developed machine learning models.

2-3-2 BELLHOP Convergence Criterion

Algorithm 1 BELLHOP, iterative Transmission Loss convergence with 1dB margin

function RunScenario(SCN(i)) ▷ Initialize with unique set of parameters
  $u' = 20000$
  $TL' \leftarrow TL(u')$ ▷ 1. Compute baseline Transmission Loss
  for $u \in \{500, 1000, ..., 15000\}$ **do**
    $TL \leftarrow TL(u)$ ▷ 2. Compute resolution Transmission Loss
    $\Delta TL(u) \leftarrow TL - TL'$ ▷ 3. Compute TL difference
    if $\Delta TL(u) \geq 1dB$ **and** $u \leq 15000$ **then**
      $u \leftarrow u + 1$ ▷ 4. Increase resolution
    else
      SCN(i) ←SCN($i + 1$) ▷ 5. *TL Converged!*
      break ▷ Load new scenario
    end if
  end for
return ($SCN(i)$, u, $\Delta TL(u)$)
end function

Another crucial factor is the definition of the convergence criterion used for estimating the minimum resolution for each scenario. The calculation runs, as presented in Algorithm 1, in an iterative for-loop, estimating the Transmission Loss TL for gradually increasing sonar resolution, measured in number of (geometric) rays u. First, a unique set of parameters, addressed in short as *scenario* $S(i)$ is selected and the problem geometry is generated. In the first iteration *baseline* TL' is estimated for an arbitrarily large number of 20,000 rays. The TL' calculated for this run is considered to be small enough that it will always provide an accurate reference value. Then the calculation is repeated for the smallest defined $u = 500$ rays and the $TL(u)$ is compared with the baseline calculation. If the difference ΔTL is smaller than 1 decibel, this resolution is considered to have good precision, comparable to that of 20,000 rays, thus it is the minimum sufficient number of rays. The loop is exited and another set of parameters $S(i + 1)$ is loaded. Otherwise, the resolution is gradually increased until a maximum value of 15,000 rays. In the later process, the scenarios that did not converge also at 15,000 rays had been discarded completely from the data, as they rise suspicion about correctness of the TL' at 20,000 rays, that cannot be easily checked otherwise. Initially the output ΔTL of each iteration was saved to a .csv file, which added up to roughly 40,000 iterations. Later it has been discovered that non-converged solutions are of little meaning to the data-based models, introducing too much noise. In effect, only the converged runs with $\Delta TL < 1dB$ were used for machine learning models. The total size of the obtained

database is 9031 entries of converged propagation scenarios. The details of the database composition are presented in table 2-2. It is important to highlight the observation about

Table 2-2: BELLHOP database size; percentage values with respect to the total number of iterations.

Total Nr. Iterations	Total Nr. Scnearios	Converged Scenarios	Non-converged Scenarios
39869	9392	9031 ($\sim$ 96%)	361 ($\sim$ 4%)

the performance of the iterative convergence test, being presumably the main reason to seek for alternatives to BELLHOP in data-driven solutions. The need of repeating the calculation multiple times for each scenario, to confirm that the calculation is accurate, makes the process very inefficient. The averaged computation time per number of rays for the obtained database presented in fig. 2-9 suggests that for the provided MATLAB script the calculation takes around 15 seconds for each additional 1000 rays. However, the baseline run takes significantly more time which can be explained by the need of creating temporary .csv files used for the spatially distributed Transmission Loss calculation. Altogether, harvesting of the data for the problem took more than 900 hours to produce just above 9000 useful samples. The above observation does not indicate that there are no better ways to speed up the algorithm (i.e. parallel processing, or rewriting MATLAB code to a more efficient language), however it gives an overview of a serious downside of the investigated acoustic model. Taking the above

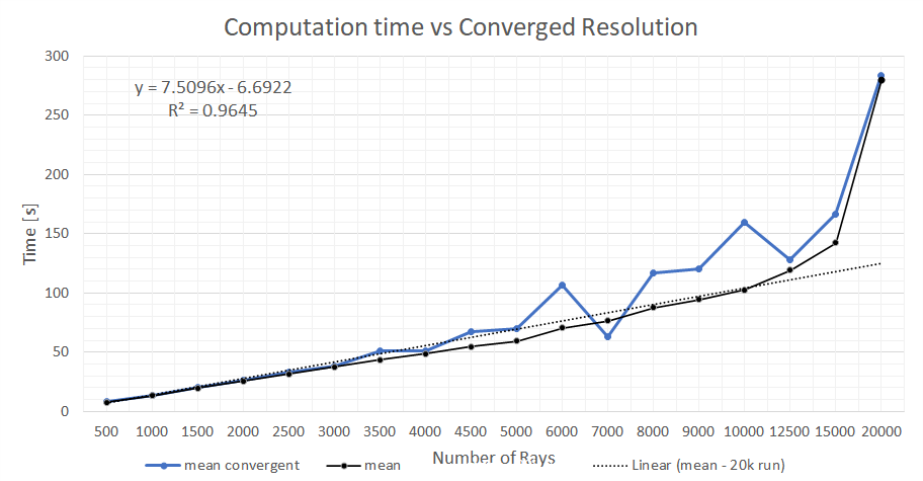


Figure 2-9: Computation time vs number of rays required to achieve convergence.

poor performance characteristics into consideration, leaves a research gap for development of an alternative solution. In the case of this research the proposed method is a data-based machine learning model. A hypothesis can be formulated, that with use of the same input parameters, a machine learning model trained on the generated simulation data, will be able to learn interdependencies between feature values and the optimal resolution (number of rays) to map the water column with required accuracy. The verification of this hypothesis is documented in the following chapters, first with use of a well-documented Decision Tree model and subsequently with an experimental Graph Neural Network.

Chapter 3

Machine Learning: Decision Tree Model

Supervised machine learning such as decision tree-based Random Forest, XGBoost or K-Nearest Neighbour algorithms has been a golden standard for solving regression and classification problems on structured data[1] for decades. There are multiple off-shelf algorithms, that are highly optimized in terms of computation performance and predictive capabilities. So it is interesting to see how well a standard ML algorithm can perform on the provided dataset and to obtain a *baseline score* - for comparison of performance with the largely experimental Graph Neural Network (GNN).

It is important to stress the fact, that obtaining the highest possible prediction score with the developed ML model is not the highest priority for this research. Rather than that, the aim is to evaluate the achievable level of inference (prediction) at the provided dataset. With this assumption, the interpretability of the results shall lead the way of the decision-making process. It also calls for minimising the variance of obtained results and by doing so, ensuring that the prediction reflects on the achievable level of generalisation in the collected dataset, rather then greedily calculated local minimum.

To investigate the problem, both regression and classification models will be evaluated for the collected dataset, which is unusual but may be argued for, after a closer investigation of level of measurements and types of the data. Also the feature representations, as it will be revealed briefly, plays a critical part in the process. Composed feature vectors will be varying in the level of complexity and added expert knowledge to reveal the most informative method. Throughout the chapter, a standard ML model development workflow will be followed. It includes a selection of the best model for a given data description, followed by a hyperparameter tuning, training and validation, and then finally prediction on the left-out test set.

[1] In this work, the structured data refers strictly to *tabular data* in contrast to unstructured data, such as images, or blocks of text, etc. usually associated with deep learning solutions.

The algorithm chosen to solve the problem is a decision-tree-based, gradient boosting model developed by Tianqi and Gusterin in "XGBoost: A Scalable Tree Boosting System" [76]. Numerous times the code will be based on resources available in scikit-learn [63], free software machine learning library for the Python programming language. There is a number of reasons supporting the choice of XGBoost as the most suitable ML algorithm for the problem, found in results of the preliminary analysis of the obtained dataset and its rather challenging characteristics, which will be explained in the section below section 3-1. The rest of this chapter contents focuses on the steps leading to the complete assembly of the Machine Learning model pipeline. The steps performed in feature engineering are described in section 3-2 with extra attention given to developed SSP identification algorithms, that have a purpose of capturing ducted modes of propagation based on the SSP vector input characteristics. This is followed by the walk-through explanation of XGBoost algorithm in section 3-3. Justification of the engineering choices such as selected loss function and evaluation metrics is in the beginning of section 3-4. Specific code description setup and details on model selection, tuning and evaluation, shortly referred to as machine learning "pipeline" can be found in section 3-4-4. The obtained results are discussed in section 3-4-7 and finally section 3-5 outlines propositions for improvements to the presented method.

3-1 Data Analysis

First thing to consider while approaching a synthesis of any data-based model is to gain a better understanding of the provided database. That may include statistical analysis of the distribution and correlation of the model features, outlier correction and other methods to investigate interdependencies in the data. In result, it should allow for a well-argued initial choice of a suitable machine learning algorithm reflecting on the characteristics of the dataset. This section explains the emerging dilemma whether the prediction of the number of rays should be considered a *regression* or *classification* problem.

3-1-1 Classification of the Target Variable Type

In BELLHOP simulation environment all model input features had to be discretized on a fixed-size grid, including the resolution, or ray number, which is the target variable to be predicted. This creates a very "clean" synthetic dataset of precisely known range, very limited variance and with no unknown noise to be filtered out. It is not necessarily making the problem easier for inference, due to limited amount of feature value combinations and thus restricted information gain. The initial choices in the definition of the simulation have an influence on how the model output was produced and what kind of acoustic behaviours were captured. For example, the ray input range was chosen to be 500 to 20,000 rays with varying step size from 500 to 2500 rays. It was not possible to evaluate the solutions with varying single-ray increment, mainly due to time needed to perform so many iterations. Thus the possibility to measure the influence of increasing resolution is limited by a step size. Also, the function space of the target variable is numerically discrete making it difficult to univocally characterize the problem as classification or regression type. Generally in machine learning the regression problems are easier to solve than classification, because continuous optimization is easier to perform than its discrete counter-part. However, under certain assumptions, it is possible for

a discrete variable to be treated as continuous, but not the other way round. To see this, let's investigate a concept of *level of measurement*, a definition used in statistics for classification of the nature of information within the values assigned to variables [54]. According to the taxonomy developed by S.S. Stevens [73] there are four levels, or scales of measurement: nominal, ordinal, interval and ratio. To review, their characteristics can be explained as such:

- **Nominal:** Unordered categorical variables, with no logical order between them. Can be either binary, i.e.: male, female or multinomial (more than two categories), i.e. colours: green, blue, red, white.
- **Ordinal:** Ordered categorical variables with a logical gradation, i.e. ratings: very bad, bad, good, very good.
- **Interval:** Numerical values (discrete or continuous) without a true zero point, hence the value equal to 0, does not mean precisely "absence" of some quantity, however the intervals between values are equal and meaningful. A good example is a Celcius scale for temperature.
- **Ratio:** Numerical values (discrete or continuous) with true zero point, and same as intervals have equal and meaningful intervals between values, i.e. a count of road accidents, or student absences in a class.

To apply this theory to the investigated dataset, it is helpful to list some observations about the domain of the target variable in BELLHOP simulation setup:

- ▷ in BELLHOP, sonar resolution n_{ray} is denoted by the number of rays, which is a discrete number, that can only take whole non-negative integer values from the domain set: $D = \mathbb{Z}^+$
- ▷ due to selection of the input values, the target variable domain is represented by a closed subset of 17 values of the original domain set, it can be denoted by: $D' = \{500, 1000, .., 5000, 6000, ..., 10000, 12500, 15000\}\,, D' \subset D$
- ▷ intervals between values of D are equal, but intervals between values of D' are not equal, both are meaningful
- ▷ set D and subset D' have a logical order
- ▷ set D does not have a logical true zero point, because without any rays the acoustic propagation cannot be calculated

This leads to following conclusions. Initially, the complete domain D of simulation resolution, represented by the number of rays, is a set of non-negative integers without a true zero point and equally spaced unit intervals. It is simply a count (of rays) or in the taxonomy presented above an interval. However, the domain of a target variable that will be used in the machine learning model is only a subset D' of the original set. Subset D' violates the definition of an interval, as it does not satisfy the condition of equal intervals between values. In result, it can be classified at most as an **ordinal variable**. In that sense, the number of rays value should be treated as an indication of the scenario difficulty to simulate with BELLHOP. It could be as well defined in a descriptive ranking such as: easy, average, hard. However, the level of discretization of the target variable is not without consequence, it reflects on

the precision of the original BELLHOP simulation. The conclusion suggests that a machine learning algorithm chosen to predict the number of rays, should be able to work with a discrete (preferably ordinal) target variable. It argues for the choice of classification over regression models, which assume that a target variable space is continuous.

3-1-2 Target variable distribution

Next, let's investigate the distribution of the target variable. As it was mentioned by the end of chapter 2 in result of a long BELLHOP simulation run, a database consisting of approx. 40,000 samples has been obtained. Data pre-processing included: first, a selection of only converged iterations, as the non-converged were found to be mostly adding noise, significantly decreasing prediction scores; secondly, a conservative correction of potentially non-converged iterations, for which 1dB margin was not reached at 15,000 rays. The latter could be considered a form of an outlier correction. After all, the useful database has been reduced to 9031 samples. This dataset reveals a heavily imbalanced, almost exponential trend in distribution fig. 3-1. Such large disproportion of the dataset cannot be ignored and to produce a reliable predictions with a classifier model, minority class oversampling techniques will be used. Adding of class-weighting factors is also discussed, yet not implemented directly. On the other hand, the regression model should not be directly affected by a class-imbalance problem as it does not assume any probability distribution of the target variable.

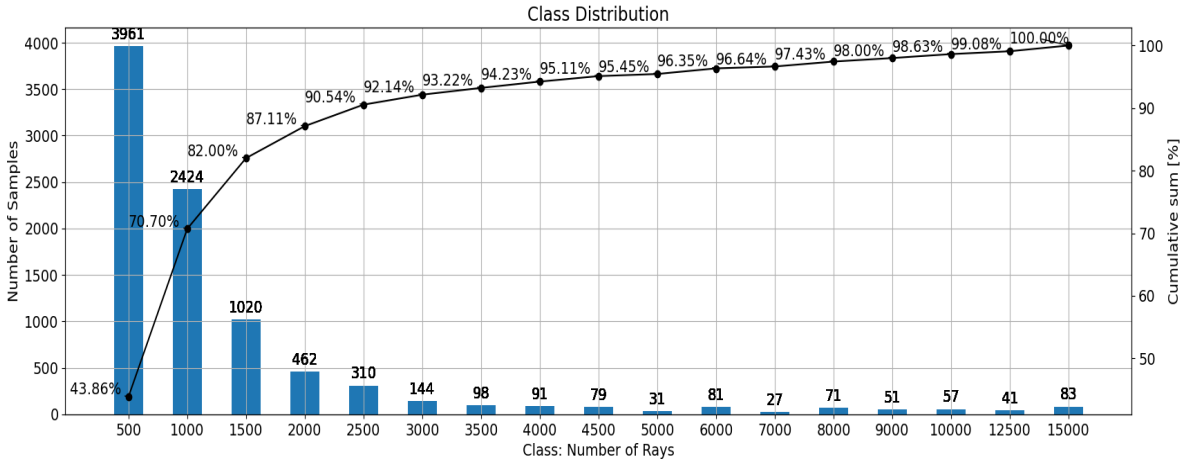


Figure 3-1: Distribution of the target variable is heavily unbalanced. Over 90% of all samples contained in the first 5 classes.

3-1-3 Resulting Approach: Classification and Regression in Comparison

The set of available features used in BELLHOP, presented in table 2-1 consists of a mixture of numerical and categorical variables. Furthermore it was deducted in section 3-1-1 that the target variable itself is a discrete ordinal value. Figure 3-1 shows the distribution of the target variable, which is highly asymmetrical, closely resembling an exponential curve.

Finding a suitable generalised linear regression model for this data, would be very difficult due to violations of linearity and normal distribution assumptions. Generalised Linear Model

(GLM) requires data transformation (scaling, normalisation) to fit the model distribution, hence also making it harder to compare with the Graph-based method for which the ultimate goal is to learn on largely unprocessed input and allow for implicit knowledge to be learned. Unlike linear regression models, the decision tree does not assume any parametric structure of the data. The data does not have to be normalized or scaled or normally distributed either. Leveraging this robustness of a decision-tree model to sample distribution, it is even possible to pose the problem as regression, by assuming the discrete target variable to be continuous.

What votes for the regression model is the extreme class imbalance and the number of classes - 17 discrete ray number values. This sets the level of complexity relatively high for any classifier algorithm, compared to i.e. a binary prediction tasks. Also, the procedures for improving classifier performance, such as class weights tuning (i.e. by PR-AUC or ROC curves [45]) to account for class imbalance, become significantly more difficult with a large number of classes. As it is not straightforward to judge, whether it will be beneficial to the prediction, it has been decided that the problem will be posed as both a multi-class classification and a regression problem, to be solved with a decision-tree based XGBoost algorithm. The best model for the problem will be revealed through a series of tests performed in parallel on classifier and regression models with varying feature vector compositions. The aim is to reinforce the understanding of how to extract important model features and effectively perform inference on the BELLHOP database.

One disadvantage of the selected XGBoost method is a questionable stability of solutions in comparison to the linear models and needs to be accounted for. XGBoost itself is a large assembly of algorithms working together to minimise the prediction error. Due to complex approach, there are numerous parameters to be tuned and also the feature selection can affect the output significantly. For that reason, the evaluation of the generalisation capability of the model shall be performed. It helps to reduce the bias resulting from algorithm instability and minimises the influence of dataset under-representation due to train/test/validation splitting. To measure the generalisation error a double-nested Cross-Validation (CV) loop is designed (see section 3-4-4) that can obtain more stable and repeatable score of the model performance. Also, comparison of classification and regression models is not readily available, due to different evaluation metrics used for training them. To allow for comparison, the regression model predictions will be rounded to the closest discrete number of rays from the original set D' and compared to classifier output in terms of per-class prediction accuracy measured in macro-averaged F1 score.

3-2 Feature Engineering

As discussed previously in section 2-3, BELLHOP input consists of 13 parameters listed in table 2-1, out of which 12 are scalar values and SSP is a single column-vector containing 24 samples of $c(z)$: underwater sound speed, which is some unknown polynomial function. The problem in which not all variables are discrete presents a mixed input type problem and can be quite challenging to solve. In machine learning it is seen more often to approach it with Deep Neural Networks [24, 82], Deep Multi-Agent Reinforcement Learning [35], which are out of scope for this research due to Graph Neural Network being equally challenging and in the main scope of interest. Standard ML solutions do not offer an algorithm capable of handling mixed input types and are designed mostly to work with structured, tabular

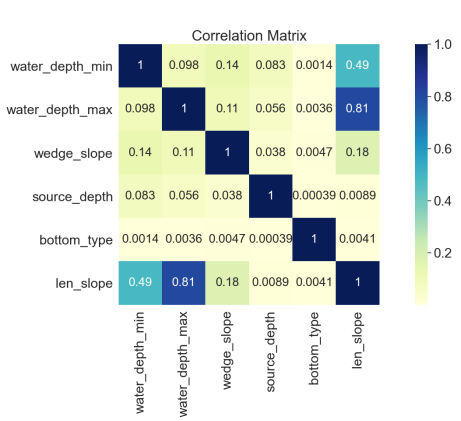
data. Therefore, majority of this section is focused on proposing methods to include SSP function into a standard one-dimensional input vector of a machine learning algorithm such as XGBoost.

However, first let us revise which features of table 2-1 will actually be included in XGBoost input. Rejecting constants and not taking into consideration the SSP input for now, only 6 parameters from BELLHOP input vary between propagation scenarios. These are $d_{src}, d_{min}, d_{max}, s, C_r, C_a$ - source depth, minimum and maximum depth of a water column, a bottom boundary slope value and two parameters describing acoustic properties of a bottom boundary. Furthermore, because C_r and C_a are coupled, thus directly correlated, they can be treated as one. It is important to note that feeding highly correlated features to any ML model should be avoided. Even thought decision-tree optimisation algorithms are not as strict, regarding features co-dependence, it is a good practice to identify and redesign or remove correlated features. Especially the direct co-dependence, that is when change in feature 1 always causes feature 2 to change at the same rate, creates redundant inputs and unnecessarily increases dimensionality of a problem. Hence, the pair of coefficients $C_r = 1.18, C_a = 0.31$ will be denoted simply as bottom type 1 (sandy) and $C_r = 0.99, C_a = 0.9$ as bottom type 2 (muddy). In effect, a bottom type becomes a categorical, integer-encoded, rather than a numerical value. That leaves only 5 numerical features, categorical bottom type, SSP input and the dependent variable n_{ray}.

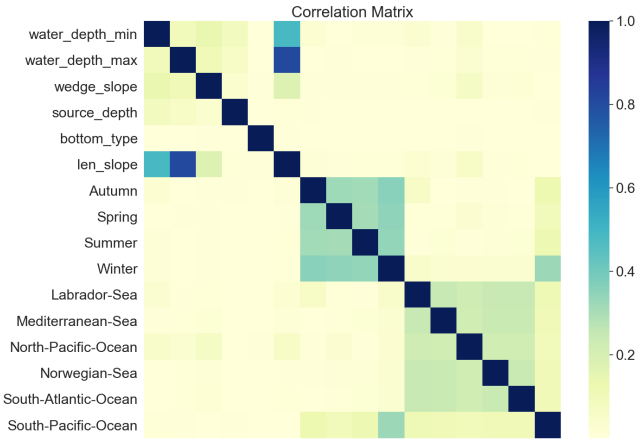
Such a low dimensionality of the data is most likely not sufficient to capture complex behaviours in domain of the target function. Searching for additional information, it has been discovered that a single feature can be added to capture horizontal geometry of a problem. Because of rules defined in BELLHOP, a lower boundary is composed of three linear segments in scenarios with a slope or a single straight segment of length $r_{max} = 44km$ in flat bottom ones. Segmentation is constrained with linear relations in the model definition, i.e. that slope segment is anti-symmetric around a mid-point, so lengths of all three are directly correlated. Only the length of inclined section l_{slope} needs to be added to a feature vector. For flat-bottom scenarios it is set to 0. Other than that, no more missing numerical parameters were identified. Altogether, the basic acoustic model parameters provides 5 basic numerical features and created categorical bottom type. From the feature correlation matrix section 3-2, it can be seen that these features are largely uncorrelated with each other. The added slope length value unsurprisingly is partially correlated with the min. and max. depth values, within acceptable level. Having just so few variables reveals how important it is to capture influence of the SSP input. To achieve that a few alternative ways will be discussed, varying in level of complexity and amount of "engineering" effort put into the feature design.

3-2-1 Alternative 1: SSP as categorical feature

First and the most straight-forward method is to treat the profile as a categorical feature with a label assigned to its location and season, i.e. "North Pacific Ocean Summer". It can be further split down into two disjoint categories for locations and seasons. To include a categorical variable in an input vector, each label needs to be encoded with a unique integer value like in case of bottom type, or by preferred in this case, One-Hot Encoding. The latter is chosen for the reason that integer encoding may suggest that labels are ordered, while there is no clear reason for that. One-Hot encoder creates binary variables in amount equal to the



(a) Correlation matrix of the six basic input features, without SSP input.



(b) Correlation matrix with one-hot encoded categorical SSP input.

number of unique labels in each class. For example, in "season" feature containing 4 labels, the encoding will result in 4 new entries: *Spring*, *Summer* etc. as presented in table 3-1. In case of "location" encoding, there are 6 unique locations, so 6 binary features will be added in the vector. As for the basic representation, a correlation matrix can be plotted in fig. 3-2b to reveal expected lack of correlation between categorical and numerical features, confirming that these inputs are independent. Some correlation is inherently captured within each of the one-hot categories. Also, a low degree correlation between "Winter" and "South Pacific Ocean" may be explained by less samples produced just for this SSP due to unexpected stopping of the simulation at one point of data gathering, which was captured in this statistical analysis.

Table 3-1: One-Hot Encoding of the season category. Labels are encoded after putting them in an alphabetical order. Analogical encoding is applied to a location category resulting in 6 new binary features.

	Autumn	Spring	Summer	Winter
Autumn	1	0	0	0
Spring	0	1	0	0
Summer	0	0	1	0
Winter	0	0	0	1

3-2-2 Alternative 2: SSP values as a set of numerical features

The alternative method is to include the actual numerical sound speed values in ML input vector. The number of points in the original data retrieved from LEVITUS database contains 24 samples until maximum depth of 1500m. This would add 24 new features, or it can be resized by interpolating original values on a different size grid. To limit redundant features for each flat-bottom scenario that does not reach d_{max} of 1500m and does not use of a full SSP vector, its size is trimmed. This also should help to differentiate between the scenarios a bit better. Nonetheless, this method has an obvious disadvantage of introducing a large number of highly correlated model variables, see fig. A-2. That can have a negative influence on quality of prediction.

Even better representation can be obtained by selecting SSP vector entries corresponding to the depth values that already appear in the feature vector for a given scenario. It means that for example in an example with $d_{src} = 200m$, $d_{max} = 1500m$ and $d_{min} = 400m$ only these three SSP values will be selected from the corresponding profile. That has the advantage of greatly reducing SSP input dimensionality. It is also deliberately enforcing some correlation between features, which may have a negative influence on model performance. On the other hand, it relieves a model of necessity to learn how to prioritize over SSP values selected from a much larger set. It could be interpreted as a small degree of an explicit knowledge that is captured in selection of sound speed values for the problem.

3-2-3 Alternative 3: SSP propagation properties

Last of the proposed representations includes a higher degree of explicitly added expert knowledge than the other two and because it requires engineering data features based on sound propagation theory discussed in section 2-1-2. There are at least three distinct sound propagation modes that can be enforced by composition of sound speed function extrema and steepness of its gradients. These three are: half-channel, surface duct and deep sound channel. Capturing these modes calls for an identification of extrema associated with Sonic Layer Depth (SLD) and Deep-Sound Channel (DSC) axis, together with deep channel boundaries. Under condition that source is located within channel bounds and that channel width is sufficient (30.48m for a surface duct fig. 2-3d), sound waves are likely to be trapped within an acoustic duct. To represent this behaviour in data features, an algorithm was designed that first identifies potential ducts at each SSP, cut to the maximum depth of a water column and then verifies if the source is located at the depth of these ducts. The algorithm consists of multiple conditional loops that verify magnitude and location of extrema at a function curve. The example output of this algorithm for a maximum length of SSP, 1500m is presented in fig. A-1. If conditions are satisfied, feature values representing channel boundaries and their averaged gradients - one for a surface duct and two for a deep channel, above and below its axis - will be added the input vector. Conditions for ducted propagation are mutually exclusive, such that only one mode or none, can be present at each time. Furthermore, same as in Alternative 2, another algorithm checks all new depth values present in the input and appends SSP values for these points only. That allows for a reduced representation of the SSP input, while extracting the most of necessary information. Using this method each scenario receives a minimum of 3 (no ducts) and a maximum of 11 new features. The correlation matrix for this complete representation is presented in fig. A-3. There it is clearly visible that this more purposefully engineered set of features introduces less severe inter-dependency especially when compared with the simple SSP vector input in fig. A-2.

3-2-4 Database Oversampling with SMOTE

On the final note, to compensate for class imbalance issue, an oversampling solution was needed. The choice of readily available Synthetic Minority Oversampling Technique (SMOTE) [20] strategy fits the needs of this project. SMOTE is a type of data augmentation method that creates new datapoints in minority classes by synthesizing values from other existing instances. It takes into account k-nearest samples close to each other (typically $k = 5$) in the features space and generates a new point. What is really useful is that the library is capable of

processing both numerical and categorical features. The samples obtained with SMOTE may have float values in the numerical feature columns, however they maintain logical categorical values as well.

3-3 XGBoost Algorithm Theory

The algorithm of choice to perform this task is the XGBoost that stands for Extreme Gradient Boosting. The term *Gradient Boosting* originates from the paper "Greedy Function Approximation: A Gradient Boosting Machine", by Friedman [50]. XGBoost is used for supervised learning problems. The algorithm is highly flexible and can solve both regression, binary and multi-class classification problems using decision tree ensembles. It is sometimes apprised in machine learning community as the one shallow learning technique that shall be mastered along with deep learning models [25].

3-3-1 Tree Building & Update

For the complete technical description of algorithm's functionality please refer to XGBoost author's paper: "XGBoost: A Scalable Tree Boosting System" by Tianqi and Gusterin [76]. This section will attempt to outline the core of algorithm functionality leading to constructing, boosting and training of tree ensembles.

To start with, it is important to understand that XGBoost builds its trees in a different way, than a standard gradient boosting algorithm does. The core of its functionality is that the model uses *decision tree ensembles* that consists of multiple Classification and Regression Trees (CART)s, rather than a single large decision tree. Contrary to a non-ensemble model, where leafs (final nodes) contain usually only error residuals, values at CART leafs represent a richer *structure score*. This score (derived in eq. (3-11)) allows for a more complex calculation of leaf weights by including a score for evaluation of a tree structure. Structure score resembles an impurity measure in a decision tree, except that it also takes the model complexity into account [76]. Ultimately, minimising the structure score enables for reduction of the prediction error and minimising the model complexity in an unified manner [22]. Consider a dataset with n examples and m independent features, for $|\mathcal{D}| = n$ a feature vector $\mathbf{x}_i \in \mathbb{R}^n$ is used to predict a dependent (target) variable $y_i \in \mathbb{R}$. As in most of machine learning models, an algorithm improves the prediction by optimizing a *regularized* objective loss function of form:

$$\mathcal{L}(\theta) = l(\hat{y}_i, y_i) + \Omega(\theta) \tag{3-1}$$

where l represents any convex, twice-differentiable function that evaluates the difference between a prediction y_i with respect to the ground truth label or value $\hat{y}_i$. Ω is a regularization term which minimizes the tree structure, along the process of optimizing trees for predictions, which due to complexity will be explained at last. Because XGBoost is an additive learner model, instead of obtaining a single tree prediction for sample x_i (which could be denoted as $\hat{y}_i = f(x_i)$), it uses an *ensemble of trees*, such that the final predictions for each sample is a sum of K multiple additive base-learners f_k. This is illustrated on fig. 3-3, see how the scores at the leafs of independently generated CARTs, are added for each x_i to obtain the

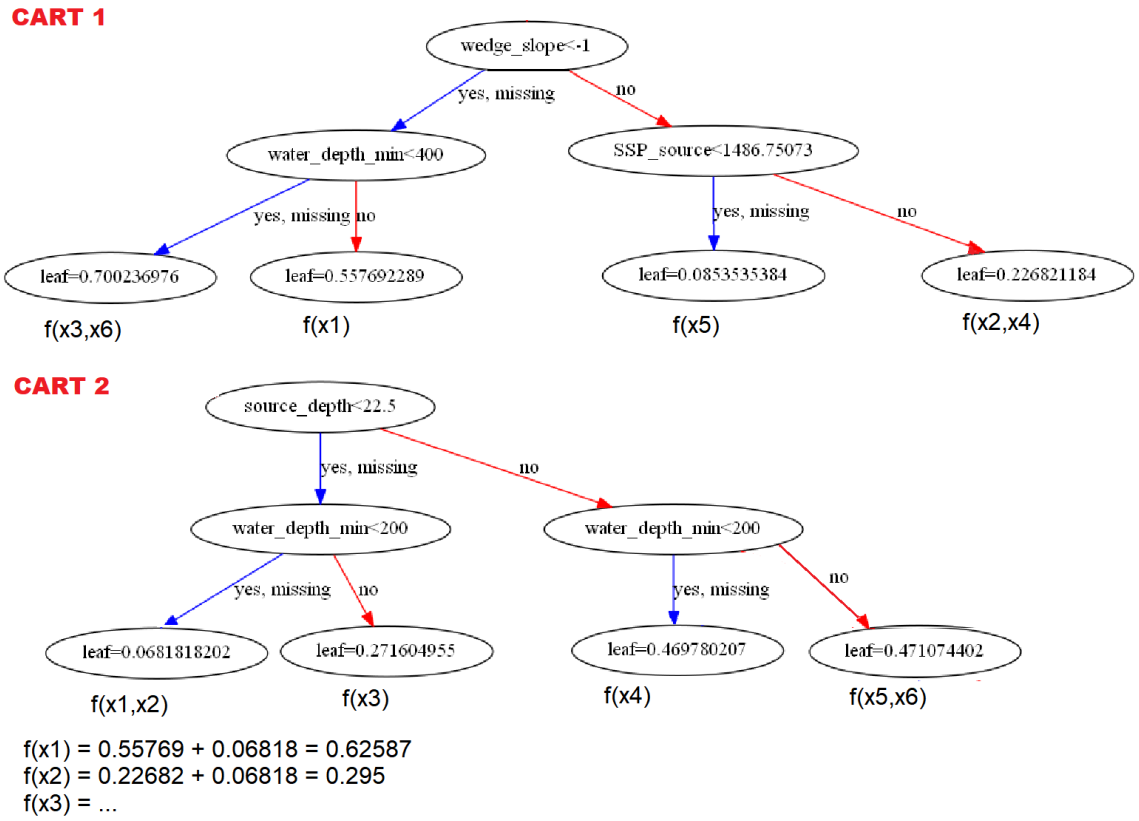


Figure 3-3: A simplified example of a CART with tree depth equal 3. In practice the estimators used for this problem are much more complex, i.e. depth 12.

final prediction score. The algorithm keeps track of indices of data points assigned to each leaf. The equation for prediction of an additive learner prediction then becomes:

$$\hat{y} = \sum_{k=1}^{K} f_k(\mathbf{x}_i) \tag{3-2}$$
$$f_k \in \mathcal{F},\ q : \mathbb{R}^m \to T,\ \ w \in \mathbb{R}^T$$

Some other important variables to describe the tree structures were introduced by the equation above. K is the hyperparameter limiting a total number of trees and T is the total number of leafs, both of them are pre-defined parameters in model setup. Each f_k corresponds to an independent tree in the functional space $\mathcal{F}$ representing a set of all possible CARTs. The tree has structure q and leaf weights w. q is responsible for decision rules (i.e. $wedge\_slope < -1$), equivalent to tree splitting on a selected feature, equivalent to mapping instances x_i to corresponding leaves of the tree. To represent a weight value on j-th leaf, w_j can be used. That weight is an average of one or more data instances $\mathbf{x}_i$ (more when multiple data instances are mapped into a single leaf, under the same a set of splitting conditions). If it is a regression model, then w_j is a real number representing error residual values of instances x_i mapped to this leaf. In the first iteration each weight is equal to a mean value of a target variable, and then it is gradually improved. The leaf value can also be negative based on a target variable.

If it is a multi-class classifier model with C classes, the leaf value represents a "raw score" for probability of the data point(s) belonging to one of the classes. The total number of trees is equal to $K \times C$ and so raw prediction for a single data instance has dimension $\mathbb{R}^C$. Final class prediction is obtained by taking a sum of all leafs in a each class-related subset of trees and mapping it between 0 and 1 using a sigmoid function. The score with a maximum probability denotes most probable class prediction. The first guess for a raw score is 0, which in a binary classification case would represent the probability being exactly 0.5. Also in this case raw leaf scores can be negative. Therefore, the objective function to be optimized for the whole ensemble of trees becomes:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} l(\hat{y}_i, y_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{3-3}$$

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda ||w||^2 \tag{3-4}$$

In place of $\Omega(f_k)$ function, L2 regularization term is added to smooth the final learnt weights and avoid over-fitting. Here L2 hyperparameter λ penalizes for large leaf scores. Additional "tree pruning" hyperparameter γ controls complexity of a tree by reducing total number of leafs w. Loss function of eq. (3-3) has functions f_k as parameters it cannot be optimized using traditional optimization methods in Euclidean space. Instead, a model is trained in an additive, greedy manner. To explain this let $\hat{y}_i^{(t)}$ be prediction of the i-th instance at the t-th iteration. The algorithm makes the optimal choice at each step as it attempts to find the overall optimal way to solve an entire problem, by adding an iteration of a CART $f_t(\mathbf{x}_i)$ that improves a model the most according to eq. (3-3).

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_i) \tag{3-5}$$

For any loss function one can find a second-order approximation in form of a Taylor expansion, that will be used to quickly optimize the objective:

$$\widetilde{\mathcal{L}}^{(t)} \simeq \sum_{i=1}^{n} [\cancel{l(y_i, \hat{y}_i^{(t-1)})} + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t) \tag{3-6}$$

In the expression above $l(y_i, \hat{y}_i^{(t-1)})$ is a known constant obtained in a previous iteration and can be removed, while g_i and h_i are defined as first- and second-order gradient statistics on the loss function:

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \tag{3-7}$$

$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \tag{3-8}$$

Because eq. (3-6) depends only on derivatives of the loss function, any twice-differentiable function can be implemented. The structure score will determine how good is a tree structure $q(\mathbf{x})$ at time t. To evaluate it at a single leaf j, define $I_j = \{i | q(\mathbf{x}_i) = j\}$ as the set of indices of data points mapped to the j-th leaf and $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$ for corresponding gradients, as illustrated in fig. 3-4. Then the final form of a structure score to be minimised

can be derived by expanding Ω:

$$\begin{aligned}\widetilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^{n}[g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T \\ &= \sum_{j=1}^{T}[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2] + \gamma T \end{aligned} \tag{3-9}$$

In this equation, w_j are independent with respect to each other and the expression inside of summation is quadratic, hence its limit can be easily found by calculating a root. The best leaf weight and the corresponding score for a fixed structure $q(\mathbf{x})$ is given by:

$$w_j^* = -\frac{G_j}{H_j + \lambda} \tag{3-10}$$

$$\widetilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T \tag{3-11}$$

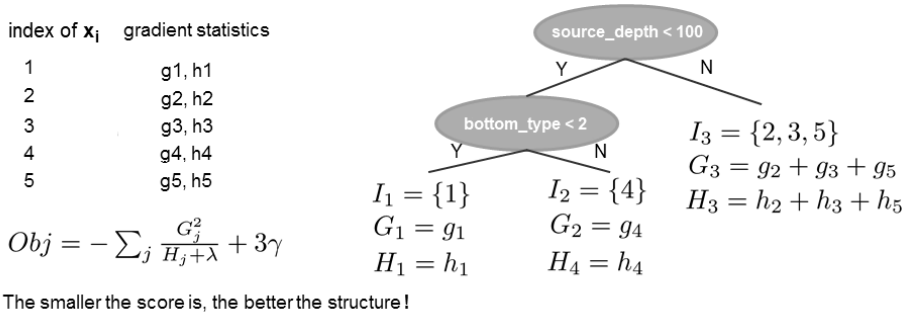


Figure 3-4: Structure score calculation for a simple tree. 5 data instances mapped into 3 leafs. Short notation I_j, G_j, H_j is used to represent data points and gradients mapped to each leaf.

Knowing how to evaluate a CART structure for a fixed $q(\mathbf{x})$ does not yet answer the question how to modify it to improve the final prediction. To estimate the best possible tree structure, ideally all combinations of splits would be created and evaluated with eq. (3-11). This is technically not possible due to combinatorial complexity of the problem. In practice, a tree is expanded one level at a time by creating a number of splits at different thresholds, i.e. $water\_depth\_min < 400$ then $water\_depth\_min < 200$. A greedy algorithm starts from a single leaf and iteratively adds branches to a tree until it reaches the maximum allowed depth. Assuming that the initial set of instances at leaf j was I_j and a new decision threshold divides it into two subsets, left and right, such that $I = I_L \cup I_R$, then the objective function eq. (3-11) *gain* due to the split can be evaluated as:

$$\mathcal{L}_{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma \tag{3-12}$$

Additional remark should be made here about the functionality of γ, which will be one of hyperparameters tuned in the later process. Appearing here in eq. (3-12), it can be seen that if the calculated gain is smaller than γ split will not be made. At the same time, in eq. (3-9), where γ multiplied by the total number of leafs increases value of the loss function, hence trees with more limited decision base-rule are favoured over more complex ones. In XGBoost this is how complexity control of a model, or *tree pruning* is performed.

3-4 XGBoost Models Tuning, Training and Evaluation

As it was already mentioned, in the proposed approach both regression and classification models will be evaluated in parallel with different features representations. In this section the details of this process will be discussed.

3-4-1 Objective Functions

Objective (loss) function influences directly the process of updating the parameters of a model during training. It is used to calculate the error residual values between observations y_i and predictions $\hat{y}_i$ that will then be mapped to leafs of an XGBoost tree using gradient statistics of eq. (3-8). The loss function has to be a convex, twice-differentiable function to comply with XGBoost requirements and to be suitable for optimization. The majority of regression problems and also the one presented here, are solved using **Root Mean Square Error (RMSE)** loss function:

$$\mathcal{L}(y_i, \hat{y}_i) = \sqrt{\sum_i (\hat{y}_i - y_i)^2} \tag{3-13}$$

Gradients used in eq. (3-8) are as simple as: $g_i = 2(\hat{y}_i - y_i)$, $h_i = 2$.

For a classifier model, the most common objective to be used is a **(Categorical) Cross-Entropy Loss** calculated on softmax output of a model prediction, resulting in a normalized probability distribution. Eeach XGBoost tree leaf $w \in \mathbb{R}^C$ (to remind: C is equal to total number of classes) stores a vector of probability scores of observations $\mathbf{x}_i$ mapped to this leaf, obtained in CART training from eq. (3-11). Then, a softmax function turns it into a normalised multinominal probability distribution vector. It assigns decimal probabilities taking values in $[0, 1]$ such that these values must add up to 1. A consequence of using softmax function is that a probability for a class is not independent from the other class probabilities. The probability of a sample $\hat{y}_i$ belonging to class c_j is given by p_{ij}:

$$p_{ij}(\hat{y}_i | c_j) = \frac{exp(\hat{y}_i)}{\sum_{j=1}^{C} exp(\hat{y}_j)} \tag{3-14}$$

calculated for each class label c_j and stored in the probability distribution vector. Then the cross-entropy loss will compare this distribution with the ground-truth label y_{ij}, where probability of the true class is set to 1 and 0 for the other classes. To put it in a different way, the target is represented as a one-hot encoded vector (binary indicator), and the closer

model prediction is to that vector, the lower the loss. The final results sums up products of all probabilities and targets in each class, for all samples:

$$\mathcal{L}(y_i, \hat{y}_i) = -\sum_{j=1}^{C}\sum_{i=1}^{N} y_{ij}\log(p_{ij}) \tag{3-15}$$

The gradient and Hessian used in eq. (3-8) of the above loss function can be approximated by: $g_{ij} = p_{ij} - y_{ij}$, $h_{ij} = p_{ij}(1 - p_{ij})$ [23].

3-4-2 Evaluation Metrics

Furthermore, it is required to choose an evaluation metrics to measure models performance. It will be used for selection of the best model, its optimal hyperparameters, validation of training iterations, and finally for a measurement of the final prediction score. This metrics should comply with the requirements of task which needs to be performed. In this case, the model should be capable of capturing characteristics of the dataset within a whole range of ray-resolution, hence prediction of samples in the under-populated tail of target variable distribution is as important as within majority classes. Once again, the class imbalance problem here is a crucial factor to be taken into account. For consistency, in following discussion the evaluation metrics *score* refers to a positive value, where higher score denotes a better performance, while *error* has exactly opposite meaning - the lower, the better. Because of the code setup, for the same metrics score and error will be used in evaluation of models, with error being simply equal to $1 - Score$. To start with, let's take classifier model as a reference. Because of the heavily skewed distribution, accuracy score ($\frac{\text{correctly classified samples}}{\text{all classified samples}}$) as in eq. (3-16) is bound to result in a positively biased evaluation of the model. The algorithm has a tendency to favour majority class predictions, simply because if in 90% of cases a prediction will be '500' or '1000', then the score will be at least at 70% (from the data distribution fig. 3-1, assuming that all these samples are identified correctly). However, this does not reflect on the goal and completely neglects higher numbers of rays. It is a misleading measure in an imbalanced domain, sometimes called an "accuracy paradox". Instead, it is proposed to use a *macro-averaged* metrics that calculates an unweighed arithmetic mean of each class partial scores, treating them with equal importance, regardless of sample size. There exist several widely accepted performance measures in imbalanced learning, all of them computable from elements of the Confusion Matrix, such as the simple one presented for a binary classification in table 3-2. There TP, TN, FP and FN represent the number of true positives (hit), true negatives (correct rejection), and false positives (type I error) and false negatives (type II error) respectively. Some of the most extensively used metrics are presented in eq. (3-17):

$$\text{Acc} = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{3-16}$$

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{F1} = \left(\frac{2}{\text{Recall}^{-1} + \text{Precision}^{-1}}\right)$$

$$\text{Recall} = \frac{TP}{TP + FN} \qquad \text{Macro-F1} = \frac{\sum_{j=1}^{c} \frac{2TP}{2TP+FP+FN}}{c} \tag{3-17}$$

Table 3-2: Confusion matrix in a binary classification problem.

	Predicted positive (1)	Predicted negative (0)
Actual positive (1)	TP (true positive)	FN (false negative)
Actual negative (0)	FP (false positive)	TN (true negative)

In information retrieval, precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned. The F1-score is a harmonic mean of precision and recall. F1-score is a special case of a larger class of F_β scores that measure the effectiveness of information retrieval, where β signifies user-defined β-times as much importance of recall-to-precision ratio [16]. F1 with $\beta = 1$ means recall and precision are equally important. According to an extensive classification metrics comparison study by Ferreri et al. [33] above metrics fall into a category of *qualitative* rather than *probabilistic* metrics. These measures are used when we want a model to minimise prediction errors, rather than assess robustness or reliability of a classifier. Using F1-score it is not possible to assess if a wrong label has been picked with a high or low class possibility. However, taking into consideration the number of classes at this stage of research, it seems to be more beneficial to maximize the ability of a classifier to distinct between different classes, thus also capturing the differences between sets of features that make ray number increase, rather than investigating sensitivity of predictions. As it is not directly possible to asses the performance of a regression model that uses continuous values, rather than discrete classes, to be able to compare these two models an additional step needs to be taken. Due to characteristics of the problem, as explained in section 3-1 classification seems to be more accurate formulation to solve this problem, hence a classifier model is taken as reference. For a regression model, predicted continuous outputs $\hat{y}_i$ are mapped to the closest discrete class label $\hat{y}_{ij}$, simply by calculating minimum Euclidean distance:

$$\hat{y}_{ij} \approx \arg\max_{c_j \in C}(\min(|c_j - \hat{y}_i|)) \tag{3-18}$$

With that approximation of discrete labels, classification metrics can be calculated for a regression model. It can be expected that due to rounding, some additional error is generated and this error is not directly modelled in the back-propagation process. It is a largely uncontrollable disproportion between predictions and the ground truth, that will remain uncorrected for now. However, in the later process it was observed that during final training iterations, using such approximation as functional validation metrics and for early stopping does not differ significantly from using RMSE values. However, it greatly slows down the process. For that reason, most of the time a regression model is evaluated with respect to its default error value RMSE eq. (3-13). Only at the final training step and for comparison of the output with a classifier in terms of per-class F1 scores, continuous predictions are rounded up.

3-4-3 XGBoost Hyperparameters

There are numerous hyperparameters that drive the algorithm of tree building and update in XGBoost algorithm. The goal of this section is to give a better overview of their influence on the process and explain the set that is selected for tuning in the grid-search procedure described in section 3-4-6. The decision about which parameters to tune was based on the

comparison of approaches found in i.a. [3]. The most important model variables for both regression and classification task can be grouped into four main categories, based on their functionality [9]:

1. Controlling model complexity:
    - `n_estimators`: total number of tree-building rounds used in a model.
    - `max_depth`: maximum depth of each tree.
    - `min_child_weight`: minimum sum of instance weight (hessian) needed in a child. If in a leaf node with sum of instance weight less than this value, then the building process will give up further partitioning. In regression that corresponds simply to minimum number of data points assigned to a leaf.
    - `min_split_loss`: a pseudo-regularization hyperparameter in gradient boosting. Loss reduction required to make a further partition on a leaf node. Equivalent to γ in eqs. (3-9) and (3-12). The larger value, the more conservative the algorithm will be.
2. Introducing random subsampling executed once for every tree constructed to avoid overfitting:
    - `subsample`: subsample ratio of training instances, i.e. setting it to 0.5 means that XGBoost would randomly sample half of training data prior to growing a tree.
    - `colsample_by_tree`: subsample ratio of columns when constructing each tree.
3. Regularization to avoid overfitting, larger values will make model more conservative:
    - `reg_lambda`: L2 regularization term on weights. Adds "squared magnitude" of coefficient as penalty term to the loss function, as in eq. (3-4).
    - `reg_alpha`: L1 regularization term on weights. Adds "absolute value of magnitude" $\alpha|w|$ to eq. (3-4). The key difference between these two is that L1 regularization reduces less important weights to zero, thus removing it completely from training. It may be used for additional feature selection in complex dataset.
4. Model specific parameters & computation performance:
    - `learning_rate`: as in every ML algorithm, learning rate controls how much to change a model in response to the estimated error, each time new model weights are calculated.
    - `objective`: technically, not a tuning parameter, but a handle to a large library of objective loss functions.
    - `n_jobs`: number of parallel CPU threads used to run XGBoost, value of -1 uses all available CPU resources.

3-4-4 Pipeline Schematic Description

Having chosen metrics used for training and evaluation, the machine learning *pipeline* can be set up. It automates all processing steps: beginning from creating and splitting different data

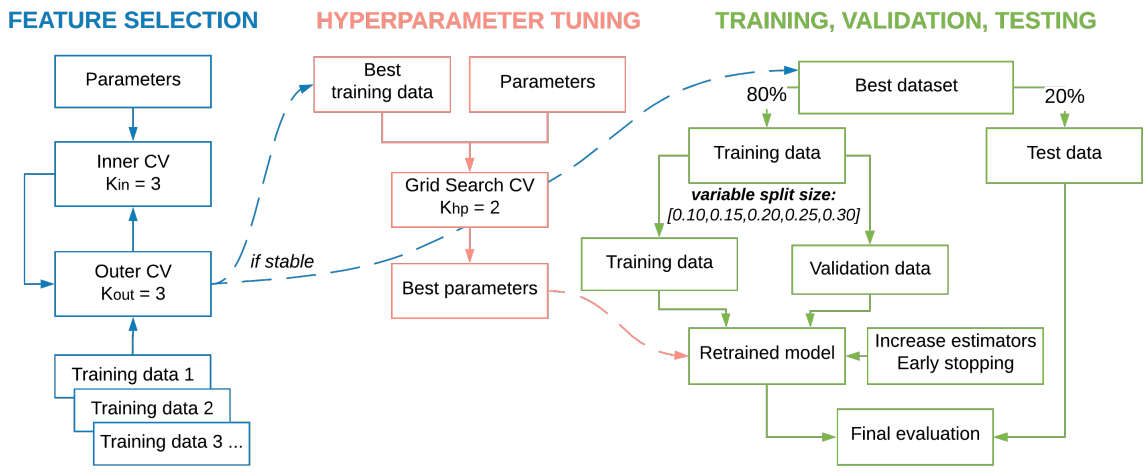


Figure 3-5: Schematic representation of the XGBoost pipeline.

representations, choice of the best dataset for a model, followed by hyperparameter tuning to find a set yielding highest validation score, then performing training of a tuned model on the whole training set and finally prediction on unseen test set. During the whole process of training and tuning only 80% of dataset is used, while the remaining 20% is used only in final prediction. The validation is performed in a CV setup but for the final one on the left out subset of the training dataset. All data splits are constrained to maintain the original dataset distribution, as in fig. 3-1, with use of Stratified-K-Split. That ensures maximum resemblance to the original set and prevents any data leakage caused by evaluating training of models on the test set, which would result in positively biased estimate of model's performance. The schematic representation of the pipeline is presented in fig. 3-5. The steps will be discussed in detail in paragraphs below.

3-4-5 Nested CV for Feature Selection & Evaluation of the Generalisation Error

The first process executed in the pipeline is a nested cross-validation, which aim is to assess an unbiased generalisation of model performance for the different feature representations discussed in section 3-2. As mentioned before, XGBoost is prone to instability due to changes in selected features and in its numerous parameters, which makes feature selection challenging, assuming that the focus is put on interpretability of the model and not crude improvement of the prediction score. Also, the provided dataset is very unequally distributed, hence the output may be biased between different splits if they are not able to fully capture characteristics of the whole dataset (not enough samples in minority classes). Nested CV fig. 3-6 consists of two CV loops, where the outer loop splits a training dataset into K_{out} - two of them used for training and one for validation - then the inner loop takes two training sets and splits it further into K_{in} parts where it measures validation score on a single $K_{in,i}$ with respect to the other two K_{in} parts and the proposed set of parameters. The process is repeated for each combination of folds, then scores across outer folds are averaged to obtain the generalisation error for the dataset and model combination. For this experiment small values of folds such as $K_{in} = K_{out} = 3$ yield generally the most stable results. Otherwise, due to few samples in

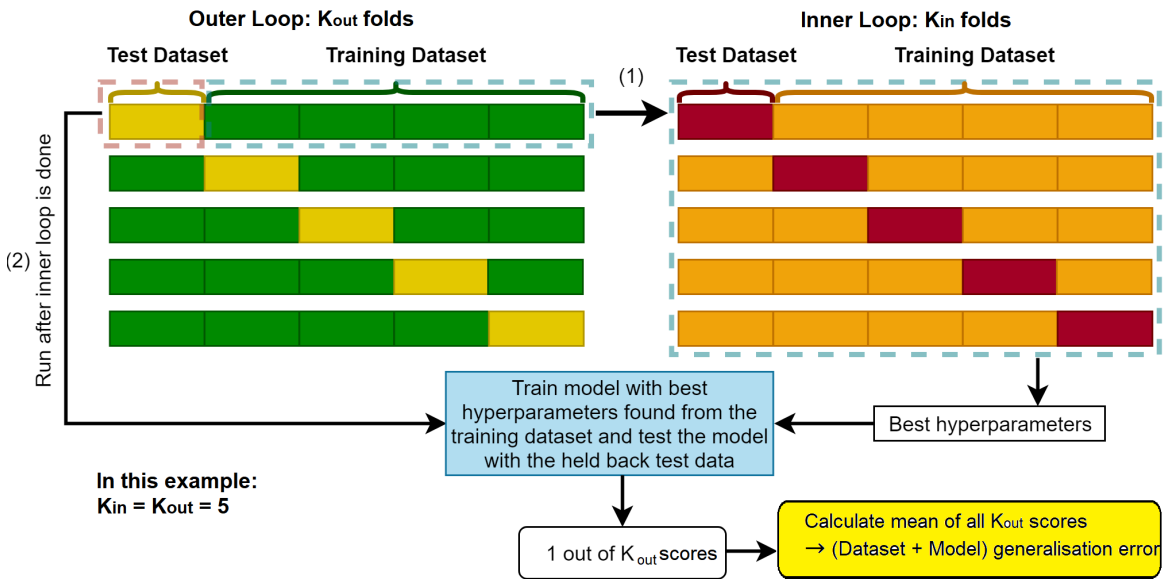


Figure 3-6: Nested CV procedure for assessment of model generalisation error.

minority classes, some of them may not be captured in folds at all. Due to computational effort nested CV is evaluated on a limited grid of parameters, presented in table 3-4, related to model complexity, and also two values for learning rate are included to rule out eventual overfitting. Total number of combinations in the parameter grid is equal to $N_{param} = 2^4$. Altogether, taking the number of folds K, datasets N_{data} and models N_{model} into account, the double loop needs to perform 1440 iterations with pre-defined number of 250 estimators:

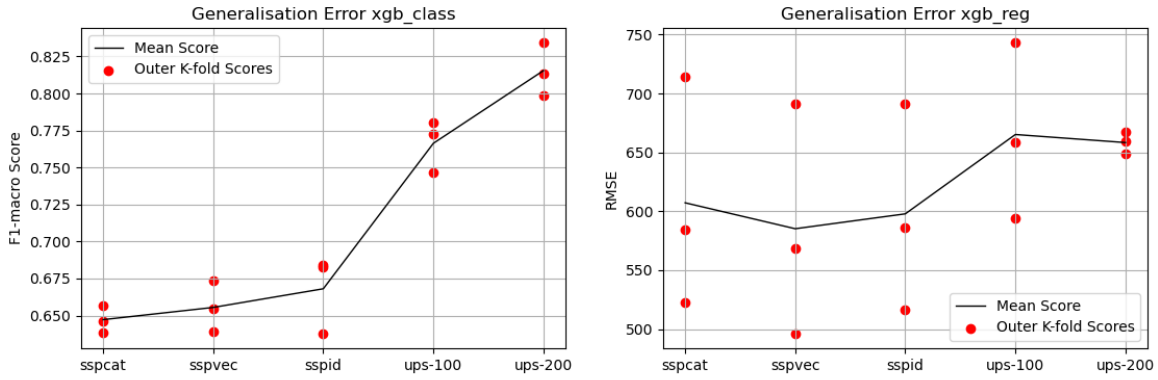
$$N_{param} \cdot K_{in} \cdot K_{out} \cdot N_{data} \cdot N_{model} = 2^4 \cdot 3 \cdot 3 \cdot 5 \cdot 2 = 1440 \quad \rightarrow \text{iterations in nested CV}$$

It needs to be underlined that this procedure is not used to estimate the best hyperparameters for the model. The inner folds may results in different sets in each split. Rather, the average score and its standard deviation is the value of interest. Following this procedure, following evaluation scores were obtained for classifier 'xgb_class' and regression 'xgb_reg' models with metrics F1-macro and RMSE respectively: In both cases adding a richer representation of the

Table 3-3: Nested CV results: outer folds mean evaluation score and standard deviations.

Dataset	SSP-cat	SSP-vec	SSP-id	SSP id-ups-100	SSP id-ups-200
Class. (F1 Score)	0.6472 ± 0.0076	0.6554 ± 0.0141	0.6681 ± 0.0215	0.7663 ± 0.0145	0.8155 ± 0.0147
Reg. (RMSE)	607.21 ± 80.06	585.51 ± 80.35	597.17 ± 72.15	665.26 ± 61.25	658.22 ± 7.67

SSP vector in sets 'ssp-vec' (alternative 2) and 'ssp-id' (alternative 3) yields a higher validation scores, than just the basic categorical encoding. Moreover, SMOTE upsampling technique 'ssp-id-ups' (alternative 4) creates a significant improvement of the score for a classifier model, while maintaining satisfactory variance within $1,5\%$. Also, the set with 200 samples in the minority classes 'ups-200' is better than the one with 100 samples 'ups-100'. For the regression model, upsampling does increases RMSE, however it should also be noted that with 'ups-200' set, the variance decreases approx. by a factor of 10. This observation suggests that for the regression model, more upsampled set should present the best generalisation property. In



(a) Classification - higher F1 Score is better.

(b) Regression - lower RMSE is better.

Figure 3-7: Generalisation error in nested CV procedure.

conclusion, for both models the best dataset is 'ssp-id-ups-200' - the SSP identification set, with additional feature columns for SSP values found only at critical depths, with minority classes upsampled to 200 samples, is the best feature representation for the problem. This dataset will be used in the next step of hyperparameter tuning.

Table 3-4: Nested CV hyperparameters.

Parameter	Values
max_depth	[8, 12]
learning_rate	[0.1, 0.01]
min_split_loss	[0.0, 5.0]
min_child_weight	[1.0, 5.0]

Table 3-5: Hyperparameters for tuning on the full grid.

Parameter	Values
max_depth	[8, 10, 12]
learning_rate	[0.1, 0.05, 0.03]
min_split_loss	[0.0, 0.5, 5.0]
min_child_weight	[1.0, 5.0, 10.0]
subsample	[1.0, 0.9, 0.8]
colsample_bytree	[1.0, 0.9, 0.8]
reg_lambda	[1.0, 3.0, 5.0]
reg_alpha	[0.0, 0.5, 1.0]

Table 3-6: Best parameters found in a 2-fold GridSearchCV.

Parameter	Class.	Reg.
max_depth	12	12
learning_rate	0.1	0.1
min_split_loss	0.0	0.0
min_child_weight	1.0	1.0
subsample	0.9	0.8
colsample_bytree	0.8	0.8
reg_lambda	3.0	5.0
reg_alpha	0.0	1.0

3-4-6 Hyperparameter Tuning

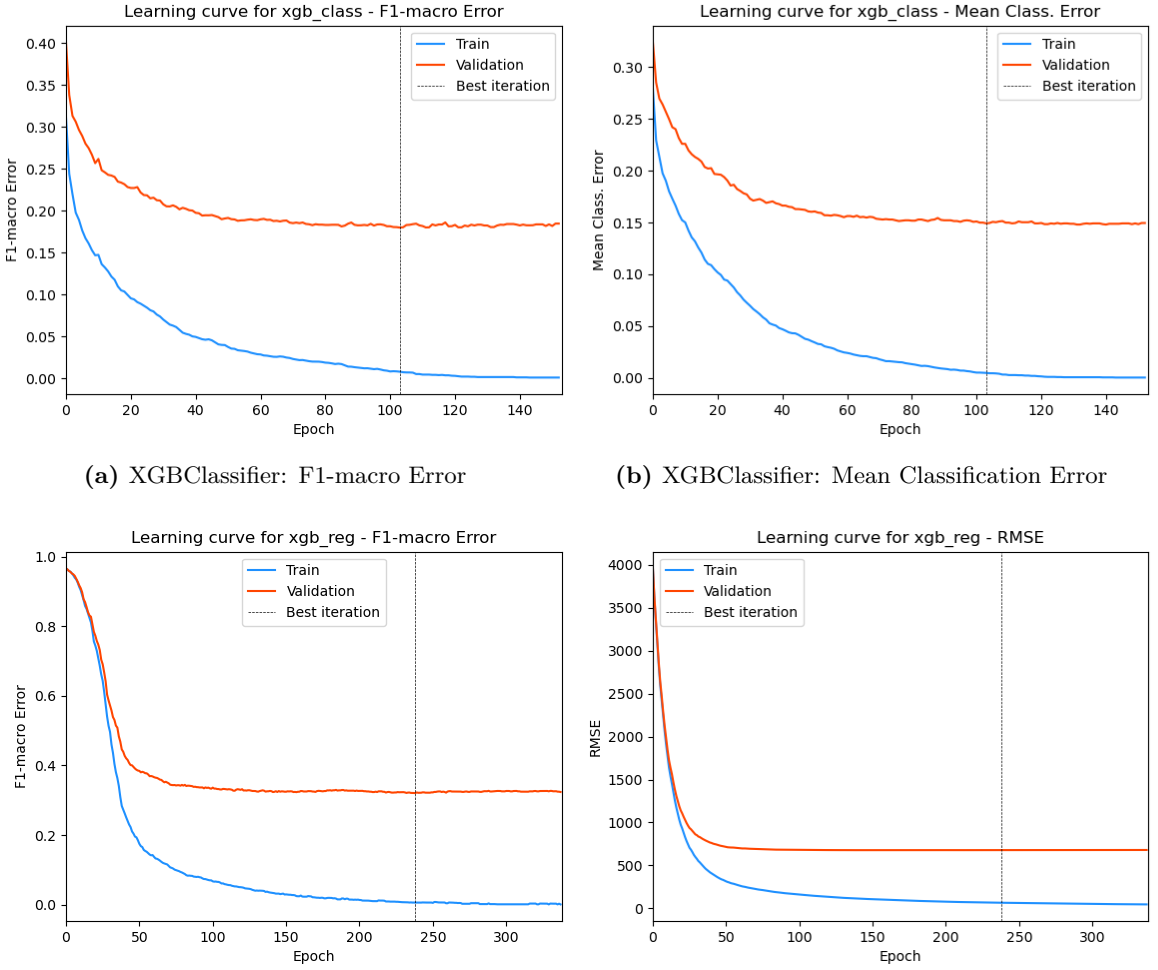
In hyperparameter tuning procedure the best dataset is evaluated in a simple stratified 2-fold CV $K_{hp} = 2$ on an extended grid of hyperparameters. This set includes all necessary variables identified in section 3-4-3. The ranges for tuning parameters were found empirically, to asses models at different complexity with maximum precision. Full tuning parameters grid is presented in table 3-5. A simple grid search is performed on the provided set of parameters and the combination with the highest validation score is being selected as best parameters for the final model. The choice for a 2-fold CV is argued by a trade-off between number of parameters to include in the grid and the number of iterations that would be performed in

additional folds. It is the most time-consuming procedure of the whole process, as the model needs to perform large number of iterations:

$$N_{param} \cdot K_{hp} \cdot N_{model} = 3^8 \cdot 2 \cdot 2 = 26244 \quad \rightarrow \text{iterations in HP tuning}$$

In result of performed grid search, the best parameters found for each model are found. They are listed in table 3-6.

3-4-7 Training, Validation and Testing



(a) XGBClassifier: F1-macro Error

(b) XGBClassifier: Mean Classification Error

(c) XGBRegressor: F1-macro Error

(d) XGBRegressor: Root Mean Square Error

Figure 3-8: Learning curves for training and validation of tuned, final models on the best selected dataset.

Before the model goes into final process of training, validation and prediction, the best hyperparameters found in previous step are applied. The number of estimators is increased to `n_estimators` = 500 and at the same time an early stopping criterion is implemented. Early stopping is triggered based on the evaluation score in the validation set. Model updates

are computed until the validation score does not improve in 100 iterations, or the maximum of 500 iterations have been reached. The choice was made not to perform another cross-validation loop at this step. However, because the final split for validation and training sets has a measurable impact on the prediction score, it is calculated as an average of 5 runs at validation set sizes varying from 10% to 30%, which should result in a less biased estimate. The influence of the validation set choice seems to be again, driven by the class imbalance issue. Looking at partial F1-macro scores in table 3-7 it can be noticed that decrease in training set size is not proportional to lower (or higher) score. Then it must be that the distribution of minority class samples between training/validation sets is the cause for score variance. It is easier to understand, when realised that if in the heavily under-sampled class like '7000 rays' with only 5 datapoints in the test set, a predictor manages to classify any of these points, per-class F1-macro score increases by 20% while the total set mean F1 automatically increases by $0.2/17 = 0.0117$ or almost 1.2%. With that distribution of the test set some variance in evaluation of predictions is inevitable. Additional evaluation metrics are also introduced at this step to help with comparison between regression and classification model output. For each of the models two error (the lower, the better) values will be registered. For classification: F1-macro Error, but also Mean Classification Error, which is the default training metrics for a multi-class classifier in XGB and closely resembles an unweighted Accuracy score. For the regressor model, beside tracking RMSE as usual, a custom F1-macro Error is implemented that makes use of output approximation as discussed in section 3-4-2. Ultimately, the predictions of two models can be evaluated. After rounding continuous output to discrete labels, classification reports can be constructed, as in tables 3-8 and 3-9, where the results are presented for the best iteration of each model. The reports allow for evaluation of per-class F1, Precision, Recall and Accuracy scores. Additionally, one can refer to the confusion matrices that are presented in Appendix A: figs. A-4 and A-5 that can give some additional insight into types of made classification errors, yet again from the classification reports: Precision and Recall scores for are roughly equal for both models, meaning that they tend to make a matching number of type I and type II errors, which is partially driven by the choice of evaluation metrics ($\beta = 1$ in Fβ score). The progress of model training and validation between the iterations are presented with learning curves in fig. 3-8, while the final prediction scores and their average in table 3-7.

Table 3-7: Final prediction evaluation scores at different validation set sizes & the mean score of all iterations. For completeness, regression model's RMSE and Rounding Error (RndErr) are also registered. The unit for these score is number of rays.

	XGBClassifier	**XGBRegressor**		
	F1-macro	F1-macro	RMSE	RndErr
Val. 10%	0.72	0.48	599.88	97.80
Val. 15%	0.69	0.47	621.29	97.03
Val. 20%	0.71	0.43	677.94	98.60
Val. 25%	0.71	0.44	591.87	99.80
Val. 30%	0.67	0.53	583.03	99.12
Mean Score	**0.70**	**0.47**	**614.80**	**98.47**

From the analysis of the results a few conclusions can be drawn. Firstly, the XGBClassifier model performs much better, both in terms of final prediction F1-macro score value and

Table 3-8: XGBClassifer: Classification Report for the best iteration at validation test size 10%.

Class	PR	RC	F1	Sup
500	0.95	0.96	0.96	793
1000	0.85	0.86	0.86	485
1500	0.73	0.69	0.71	204
2000	0.61	0.72	0.66	93
2500	0.65	0.55	0.60	62
3000	0.68	0.72	0.70	29
3500	0.57	0.40	0.47	20
4000	0.84	0.89	0.86	18
4500	1.00	0.88	0.93	16
5000	0.80	0.67	0.73	6
6000	0.75	0.94	0.83	16
7000	0.75	0.60	0.67	5
8000	0.69	0.64	0.67	14
9000	0.69	0.90	0.78	10
10000	0.82	0.82	0.82	11
12500	0.38	0.62	0.48	8
15000	0.78	0.41	0.54	17
Macro Avg.	**0.74**	**0.72**	**0.72**	1807
Accuracy	**-**	**-**	**0.85**	1807

Table 3-9: XGBRegressor: Classification Report for the best iteration at validation test size 30%.

Class	PR	RC	F1	Sup
500	0.93	0.85	0.89	793
1000	0.72	0.78	0.75	485
1500	0.58	0.62	0.60	204
2000	0.44	0.55	0.49	93
2500	0.54	0.42	0.47	62
3000	0.41	0.48	0.44	29
3500	0.42	0.40	0.41	20
4000	0.61	0.61	0.61	18
4500	0.75	0.75	0.75	16
5000	0.33	0.50	0.40	6
6000	0.79	0.69	0.73	16
7000	0.40	0.40	0.40	5
8000	0.77	0.71	0.74	14
9000	0.55	0.60	0.57	10
10000	0.40	0.73	0.52	11
12500	0.25	0.38	0.30	8
15000	0.00	0.00	0.00	17
Macro Avg.	**0.52**	**0.56**	**0.53**	1807
Accuracy	**-**	**-**	**0.74**	1807

decreased variance between iterations in comparison to the XGBRegressor. Smaller variance of the classifier could have been already observed in the nested CV procedure while comparing outer fold scores fig. 3-6 for the two models. Just the fact that regression could not classify any of the 17 points in '15000' class lowers its classification metrics score by more than 5.8%. Secondly, from the learning curves a substantial difference between training and validation scores is noted. While the training curve (orange line) converges to 0 rather quickly, the validation curve (blue line) is stagnating at a certain value and will not diminish even for an increased number of estimators. Also, there is little to none overfitting present. Such output usually suggests that a validation set does not fully capture characteristics of the function space to predicted. Simply speaking, may contain points in the minority classes that have a set of features that is not present in the training set, and thus a model has very little chance to learn these. On the other hand, where the number of samples is large, as in the first 3 classes, the predictors produce generally satisfactory results, with the classifier predicting 96%, 86% and 71% points in these classes correctly. It can be seen that due to training set upsampling there is also a visible disproportion between best validation scores (marked as best iteration in fig. 3-8) and the prediction scores. The difference can be measured directly, so for example F1-metrics is roughly 0.1 or 10% lower in prediction than it is in validation. The obtained results, favour classification approach over regression.

3-5 Proposed Improvements for Increased Interpretability of Solutions

From attempts to predict the resolution of BELLHOP model, based on the original simulation input and even directly from the initial analysis of collected database, it became clear, that the problem poses certain unfavourable characteristics, making it difficult to solve with Machine Learning algorithms with high precision. The most consequential of them is supposedly lack of balance in obtained data samples, with these connected to high required number of rays being in severe minority. Based on data distribution, only approx. 9.5% of all scenarios fall in the range of resolutions above 2500 rays, which yet corresponds to 12 out of 17 possible discrete output values. The disproportion is highly dependent on pre-selected, relatively sparse, input grid used in the BELLHOP simulation. Nevertheless, the assumption was made that no more data to equalize dataset distributions can be produced. This decision has driven the effort to focus on the interpretability of results and to capture, as closely as possible, the influence of separate inputs, or their combinations, that could lead to increase in the number of rays.

Plotting target variable partial dependencies: To analyse this dependency a method from the interpretable machine learning toolbox was chosen: Individual Conditional Expectation (ICE) plots proposed in [38]. ICE plots display one line per target value instance that shows the instance's prediction changes when a feature changes [2]. This allows for displaying a human-readable representation of the designed model relationship with input features. The plots are obtained by calculating estimated conditional expectation curve:

$$\hat{f}_S^i(\mathbf{x}_S) = f(\mathbf{x}_S, \mathbf{x}_{C_i}|\hat{\theta}) \tag{3-19}$$

where and $f(\mathbf{x}|\hat{\theta})$ denotes a classifier, that takes splits the model features into two complementary subsets, investigated feature: $S \in 1, .., m$ and "all others": $C \in 1, ..., m$ S. Then $\mathbf{x}_S$ is the evaluated feature value that will be varied within the range of possible input values, while x_{C_i} is kept constant. Finally $\hat{\theta}$ is vector of weights of the model. Similarly, one can plot a Partial Dependence Plot (PDP) for the overall effect averaged over all data samples in N. It is equivalent to say that the average of ICE plots in a dataset is the corresponding PDP for the selected feature [5]. Unlike partial dependence plots, ICE curves can uncover heterogeneous relationships available in the data, while PDPs can show what an average relationship between a feature and the prediction looks like. This only works well if interactions between features for which the PDP is calculated and the other features are weak [84].

Without further ado, the ICE plots of all numerical model features (excluding one-hot-encoded SSP categories), are presented in fig. A-6. The overall plot is too large to be presented here and the resulting relations, but for one discussed here are mostly chaotic, thus proving that a combination of numerical features and/or SSP input, rather than a single feature drive the output value. Nevertheless, ICE plots were able to capture an important observation, to present this let's analyse the output for 'wedge_slope' (bottom boundary inclination feature) which ICE plot is visible in section 3-5.

It can be quickly realised that most of samples in the dataset follow a visible linear relation between output and slope values, which average trend is captured by the PDP plot (blue line). It is logical to assume that the resolutions required for slope-0 scenarios, corresponding to

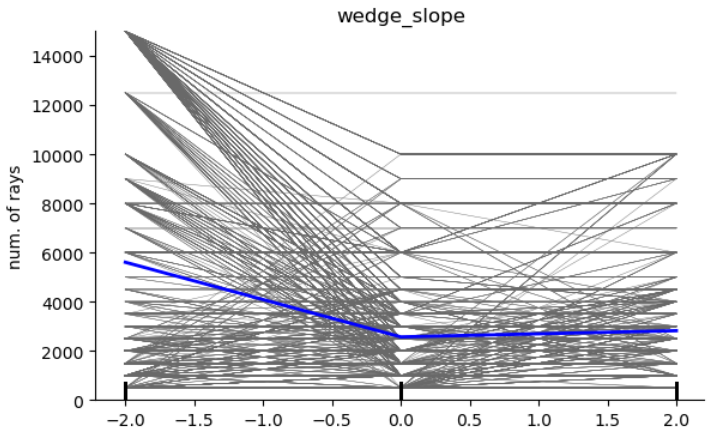


Figure 3-9: ICE plot for varying 'wedge_slope' value. PDP plot is marked in blue. It is a mean of all grey lines.

easier, straight water-column propagation cases are lower. It can also be confirmed by plotting the target variable distributions of data subsets split on the slope value as in fig. 3-10.

Each of these three subsets follows a slightly different distribution with slope-0 being the most distinct, because majority of samples fits in classes up to 2500 rays. It is fair to assume that for this subset of approx. 3000 samples, classes with just a few samples, quite likely are result of the simulation environment instability and could be treated as outliers, narrowing the target range to the first 5 classes. For the subsets with positive and negative slope distinction is yet not so clear, however it is worth to notice that all but one 12500, and all 15000 rays scenarios were found in the cases with declined bathymetry. They create a heavy distribution tail, in comparison to more equalised slope-2 subset.

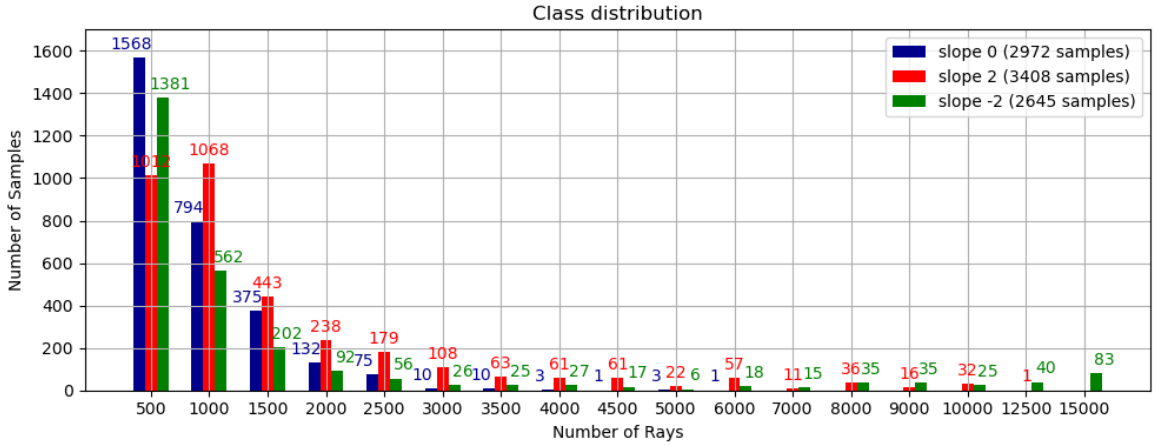


Figure 3-10: Target variable distribution in the subsets split on slope value.

Making predictions in subsets of data: This observation motivates an attempt to divide the problem into smaller sub-problems by splitting the dataset on slope feature value. It separates

the straight channel propagation, however it was also proven to be beneficial to separate the declined-bottom from inclined-bottom cases. Furthermore, an outlier correction procedure rather than SMOTE is applied to the classes with $\leq 1\%$ instances. The samples within these classes are first re-labelled, to propagate them to the next higher class (which is a much preferred solution over propagating to lower classes; considering the practical application of the developed software mapping the environment at higher resolution can impact performance, but mapping with an insufficient ray number may lead to detection or navigation errors) and tested, if in combined higher class is above the one-percent threshold. This procedure can be repeated only up to two classes above. If the condition is fulfilled a correction is valid, but otherwise the data from heavily under-sampled class are treated as outliers and removed completely. That slight modification impacts only a small fraction of all data, but it allows for a dropout of a few unrepresentative classes in each subset, in turn making the problem much easier to solve. The process of XGBClassifier Hyperparameter (HP) tuning, training and validation can be repeated for the smaller, yet more consistently distributed datasets. The results show a significant improvement of the final prediction scores for two out of three partials datasets, with an average score of all three models equal to 0.78 in F1-macro metrics. The declined-bottom (slope at -2 deg/m) subset poses the greatest difficulty to classification, with the F1-macro score dropping to 0.64 just for this subset. It may suggest existence of some highly non-linear behaviour in the data (i.e. due to results instability), however it is also worth to note that it is the smallest of all three subsets with 2645 samples, where the positive slope set with similar distribution has 3408 samples. It is also, not without a reason, the subset for which the best F1-score could have been achieved. More results of each "sub-model" training, such as learning curves and correlation can be seen in the appendix A.

Table 3-10: Prediction scores in the data subsets, separated on the slope value.

Dataset	Slope 0	Slope 2	Slope -2
F1-macro	0.81	0.89	0.64
MEAN SCORE		**0.78**	

The approach yields a satisfactory improvement of the prediction scores and more importantly, it does allow for a better understanding of trends present in the collected dataset. Nevertheless, it comes with a price of a decreased prediction resolution, because due to out-of-distribution instances correction, some classes are removed completely for each subset and a model.

3-5-1 Recommendations & Conclusions

Due to time-constraints and priority of the project given to GNN design, any further ensemble methods such as One-vs-Rest logistic regression models were not developed. Measurably better results obtained with classification models suggest that an ensemble model could achieve a better performance on a heavily imbalanced dataset. However, to implement such a solution an additional effort would need to be put in tuning the class weights with PR-AUC or ROC curves to correct for the sample occurrence frequency. This can be both difficult when 17 classes are considered, and very time consuming regarding the tuning and training process, as

it requires tuning of not one but multiple different models, one for each class. This approach could be taken even further by developing an ordered logit model for an ordinal regression task first proposed in [34]. An ensemble model of this type has a capability of accounting for the natural order of the target variable by executing an sequence of binary classification tasks. From a learning point of view, the ordinal structure of dependent variable is an additional information that a learner should try to exploit. Solutions like this were successfully developed for ranking systems in for example [34, 27, 18, 49]. Finally, an obvious improvement could be made, if new data was generated with BELLHOP parameters set for the "difficult" propagation cases to fill up gaps in distributions of the target variable. To help with selection of the simulation parameters, one could refer to the observations made during analysis of XGBoost results, to which some additional BELLHOP-specific remarks are made in the final chapter of this document.

Chapter 4

Deep Learning: Graph Neural Network Model

The content of this chapter describes an attempt to design a Graph Neural Network (GNN) for the BELLHOP problem. Graph Neural Networks first proposed by in a paper by Scarselli et al. in *"Computational capabilities of graph neural networks"* [71] fall under a category of machine learning solutions that exploit an explicitly stated structure of data representation. Other examples of data with a defined relational data structure are social networks, parse trees, chemical graphs, road networks, and physical systems with known interactions [15]. Recently, more models arriving from an intersection of structured data and deep learning have been developed. Some examples include: lower-dimensional entity embeddings [78] and representation learning [81], graph networks learning heterogeneous knowledge [80], geometric learning [17], non-local neural networks [57] and different approaches to a design of GNN architectures [71, 37, 53, 14, 15]. All of the above methods have a common denominator - by computational reasoning not only over data entities, but also a relational structure that joins them, they learn to approximate the target function space from the high-dimensional representation the feature space. Possible applications may include tasks such as attribute value prediction, but also completion of a graph structure by adding new inferred nodes, connections and even whole subgraphs [75].

The chapter structure is as follows. In section 4-1 the theoretical basis for graph-based computations is presented together with the necessary nomenclature, equations and notation in section 4-1-3, that will be used in the report. Different possibilities of combining graph elements into neural architectures are presented in section 4-1-4 with special consideration to the Message Passing Neural Network in section 4-2, which is the architecture type implemented in the default KGCN design. Then section 4-3 gives an overview of Grakn [75], which is the main tool for graph database management used in this project. Further, starting from section 4-4 containing the description of the model ontology design process, the reporting focuses on integration of all above concepts into one functional GNN architecture. The explanation becomes more code-based as the solutions designed there have a great impact on the performance of the model. Proposed design for the Grakn's Knowledge Graph Convolutional

Network (KGCN) uses multiple modules and code libraries interconnected into a large network. Understanding its interfaces is crucial for the integration of the method. Section 4-5 contains the schematic descriptions and explanation of the overall process starts at retrieving sample graphs from the schema, and passing them into the MPNN module, through a series of encoder, embedder and decoder networks trying to find the best numerical latent feature representation for the underlying ontology structure. Finally the intermediate results are presented in a series of the tests in section 4-6 and briefly summarized in section 4-7.

4-1 Deep Learning over Knowledge Graphs

4-1-1 Knowledge in Machine Learning

Before diving into technical description of the knowledge-graph-based learning algorithms, let's take a brief look at what it means to learn the "knowledge" in a context of a machine learning model. The concepts that were already introduced in section 1-1 will be extended here with a practical application to a Graph Neural Network model design.

In philosophy and theory of knowledge, one often refers to knowledge as "justified, true belief" standing in contrast to "revealed" fact, or an information. What makes human intelligence superior over any kind of existing machine learning algorithm is the ability of learning and logical reasoning over the stored knowledge-base (human memory). One example to visualise this capability is to consider the structure of any formal language. Having access to a limited number of words, and obeying a finite number of grammatical rules, human is capable of conveying an immense number of messages - this idea is one of foundations of Noam Chomsky's theory of language [26] - describing (with reference to von Humboldt's description [77]) a linguistic system that makes "infinite use of finite means". Knowledge encapsulates more than just facts, as it requires *understanding*, gained through experiencing, associating, and *cognition*. It builds on reasoning and awareness, rather then plain observation. Taking above into consideration, a broader reach and increased complexity of knowledge representation is revealed, compared to plain data, which could be expressed as:

$$Data + Context = Knowledge$$

Taking above into consideration, much broader reach and increased complexity of knowledge, when to compared to tabular data is revealed. Relational reasoning over data, which involves manipulating structured representations of entities and relations, needs a suitable representation of the *context* to express the additional layer of complexity between the data instances. Here it takes form of a multi-dimensional graph.

4-1-2 GraphNet [15] Method Introduction

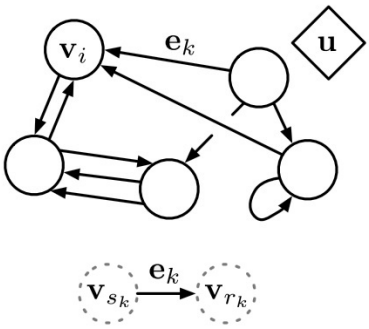
To understand the inference potential of graph-based deep learning solutions, first the theoretical background of the method will be investigated. The ideas presented here are largely the result of a publication by Google DeepMind team *"Relational inductive biases, deep learning, and graph networks"* [15] whose work was adapted by Grakn [75] in the recently developed deep learning graph network called KGCN. To avoid extensive mathematical derivation, the explanation below is limited to the most applicable part of the graph learning theory with emphasis on solutions tested during the project.

On importance of relational inductive bias: To begin with, a fundamental concept of the investigated method - the *relational inductive bias* - will be introduced. In essence, due to the form of specific architectural assumptions encoded in a graph structure, allows to guide the model towards generalising over explicit knowledge-based relations, rather than only statistical correlations, that can be (sometimes falsely) identified in the raw data. By creating an edge-to-edge connection between the two nodes, introduces a strong assumption on interaction between these entities. It allows to prioritize explicitly related features over the ones that are missing links in the graph representation, independent of the observed data [62]. In a larger picture, the bias goes beyond the GNN architecture and can be identified in a variety of neural networks components. Also, in the context of previously developed Extreme Gradient Boosting (XGBoost) model the inductive bias was introduced by the composition of the feature vector or even a selection of the optimal tree structure. In a Bayesian model, inductive biases are typically expressed through the choice and parametrization of the prior distribution [41]. In other examples an inductive bias might be a regularization term [60] added to avoid overfitting, or it might be encoded in the optimisation algorithm itself. Inductive biases often trade flexibility for improved sample complexity and can be understood in terms of the bias-variance trade-off [36]. Ideally, inductive biases improve the search for solutions without substantially diminishing performance, as well as help finding solutions which generalize in a desirable way. However, mismatched inductive biases can also lead to suboptimal performance by introducing constraints that are too strong [15]. In the case of investigated BELLHOP database, that has a very limited feature space, an assumption can be made that the relational inductive bias may be helpful for a GNN to reach a level combinatorial generalisation that could not me achieved without a relation-centric model.

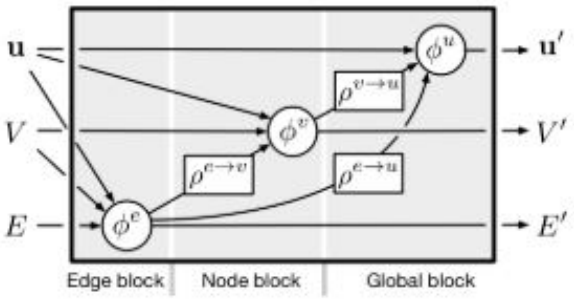
Developed GN methodology has been realized in a practical implementation of GraphNet open-source Python library, based on TensorFlow (TF) and Sonnet [61]. The library collects and extends various graph neural network designs, i.a. Message Passing Neural Network (MPNN) [71, 14] and Non-Linear Neural Network (NLNN) approaches [37, 79]. The main module for computations in the proposed framework makes use of *GN-blocks* that can facilitate a range of adjustments in terms of within-block structure. Each GN-block calculates an aggregated update of real-valued graph attributes. GN-blocks can be connected to each other, allowing for composing complex architectures much like i.e. a deep neural network is composed of convolutional, fully-connected and pooling layers. This is achieved thanks to "graph-to-graph" feed-forward process, which takes a graph G as input, performs computations over the structure, and returns a graph G' as output. Output result is compared against a target (graph, node or numerical value), such that correctness of obtained predictions can be measured with use of a differentiable loss function. As in a standard NN model gradients of a loss function are backpropagated to the GN blocks to update their parameters and gradually improve the accuracy of prediction.

4-1-3 Mathematical Formulation of GN-block Functions

Graphs, being explicitly structured databases, are representations that naturally support pairwise relational structures, or even multi-node relational structures (then called a hypergraph). Graph is constructed from nodes, edges, relations and attributes. Relations are expressed using binary statements, called triples, that connect exactly two nodes with a single relation. Usually the relation has a specified direction, then elements of a triple can



(a) Example of a GN-block. A node is denoted as $\mathbf{v}_i$, an edge as $\mathbf{e}_k$ and global attribute as $\mathbf{u}$. In directed, one-way edge message transfers from a sender with index s_k to a receiver r_k node through edge $\mathbf{e}_k$.



(b) Schematic representation of a fully-connected GN block update. Update functions ϕ are calculated in the presented order on ρ aggregated sets of elements the graph.

Figure 4-1: Basic computational unit in a GraphNet framework is a fully-connected GN-block. It consists of there update functions ϕ and three aggregate function ρ.

be further denoted as: *sender* (object) and *receiver* (subject) joined by *relation* (predicate). Arbitrary number of attributes can be assigned to nodes and edges of the graph. They can take arbitrary formats varying from string or numeric sequences and sets, image tensors to real-valued vectors. It is also possible to define a global attribute that affects all edges and/or nodes without any direct edge-to-edge relation. For the rest of the report, when a "graph" is addressed, it should be understood as a directed, attributed multi-graph with a global attribute, like the one depicted in fig. 4-1a (unless stated otherwise).

The core unit of computation is a *fully-connected* GN-block depicted in fig. 4-1b, that consists of six functions: three *update* functions ϕ for edges, nodes and a global attribute, and three corresponding *aggregation* functions ρ, which add/reduce a set of outputs from multiple graph elements. In mathematical notation, a complete graph can be denoted as 3-tuple $G = (\mathbf{u}, V, E)$. $V = \{\mathbf{v}_i\}_{i=1:N^v}$ is a set of nodes of cardinality N^v, where each $\mathbf{v}_i$ is a node's attribute. $E = \{(\mathbf{e}_k, r_k, s_k)\}_{r_k=i=1:N^v,\ k=1:N^e}$ is the set of edges of cardinality N^e, where each $\mathbf{e}_k$ is the edge's attribute, r_k and s_k denote an indices of receiver and sender nodes $\mathbf{v}_{r_k}, \mathbf{v}_{s_k}$. Finally $\mathbf{u}$ is a global attribute that affects the update of each and every node and edge in graph. The equations used in the update of a fully-connected GN block are presented above in a pseudo-algorithm that gives a good overview of the order of functions being executed. They are also provided here:

$$\begin{aligned} \mathbf{e}'_k &= \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) & \bar{\mathbf{e}}'_i &= \rho^{e\to v}(E'_i) \\ \mathbf{v}'_i &= \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}) & \bar{\mathbf{e}}' &= \rho^{e\to u}(E') \\ \mathbf{u}' &= \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}) & \bar{\mathbf{v}}' &= \rho^{v\to u}(V') \end{aligned} \tag{4-1}$$

In the default order of calculations: first, the edge update ϕ^e is applied per edge to calculate new attribute values $\mathbf{e}'_k$ for all edges that project to the same (receiver) node denoted by index $r_k = i$. Set of updated per-node edge attributes $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,\ k=1:N^e}$ is aggregated per-node by $\rho^{e\to v}(E'_i)$ to compute the aggregated update $\bar{\mathbf{e}}'_i$. The second of edge aggregate functions $\rho^{e\to u}(E')$ can be applied directly after all edges were updated. It is computed on a

Algorithm 2 Steps of computation in a full GN block.

function GRAPHNETWORK(E, V, $\mathbf{u}$)
  for $k \in \{1 \dots N^e\}$ **do**
    $\mathbf{e}'_k \leftarrow \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u})$ ▷ 1. Compute updated edge attributes
  end for
  for $i \in \{1 \dots N^v\}$ **do**
    let $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,\, k=1:N^e}$
    $\bar{\mathbf{e}}'_i \leftarrow \rho^{e\to v}(E'_i)$ ▷ 2. Aggregate edge attributes per node
    $\mathbf{v}'_i \leftarrow \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u})$ ▷ 3. Compute updated node attributes
  end for
  let $V' = \{\mathbf{v}'\}_{i=1:N^v}$
  let $E' = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$
  $\bar{\mathbf{e}}' \leftarrow \rho^{e\to u}(E')$ ▷ 4. Aggregate edge attributes globally
  $\bar{\mathbf{v}}' \leftarrow \rho^{v\to u}(V')$ ▷ 5. Aggregate node attributes globally
  $\mathbf{u}' \leftarrow \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u})$ ▷ 6. Compute updated global attribute
  return $(E', V', \mathbf{u}')$
end function

set containing all updated edges across all nodes $E' = \cup_i\{E'_i\}$ and results in an aggregated global edge update $\bar{\mathbf{e}}'$. Next ϕ^v is applied to compute updated node attributes $\mathbf{v}'_i$ in each node. It takes the node value from the previous iteration, the global value, and aggregated per-node new edge values calculated in the previous step. The resulting set of node outputs is $V' = \{\mathbf{v}'_i\}_{i=1:N^v}$ is then aggregated with $\rho^{v\to u}(V')$ into reduced representation of all node updates $\bar{\mathbf{v}}'$. Finally, ϕ_u is applied once as the global output is calculated, taking updated and aggregated values of all other components and a global value from the previous iteration.

The flexibility of modification of each ϕ function in Equation (4-1) allows for a GN-block to be tailored in terms of its input arguments, as well as, in what format each node, edge or global update output is produced (i.e. vector size or magnitude). In essence, each ϕ must be implemented with some function, which argument signature determines the information that is required as input. Each update function is a stand-alone model that learns to represent a function space associated with its designated graph element. Due to the fact, that node and edge update functions ϕ are reused between all individual graph objects of a given type, makes the GN method convolutional and reinforces *combinatorial generalization* property.

On importance of graph permutation invariance: It is important for ρ functions, which take a set of graph elements as input, to be invariant to permutations of their inputs and to take variable numbers of arguments. The reason for that comes from arguably the most important theoretical assumption in the whole GN theory. It states that a graph is a data representation of a system, which order is naturally undefined or irrelevant. For that reason it is also invariant to changes in the order of its elements. Therefore, it is a necessary condition for a deep learning algorithm, designed to perform *relational reasoning over graph databases*, to reflect on the natural graph property of *permutation invariance*; unless, in the face of constraints imposed by an explicit relational structure [15]. Essentially, for any two graphs which are constructed from the same set of entities and connected by pairwise relations in the same configuration of triples, there is a structural equivalence which should be captured by a GNN algorithm, no matter what "shape" they are presented to the model. Here, the use

of word "shape" an illustrative way to describe the order of feeding in arguments to the GN-block aggregate functions, as they scan the elements of the graph and aggregate the attributes. The example describes precisely what is known as *isomorphic property* of the structure. Two graphs are said to be isomorphic, if their vertices can be mapped into by a bijective function. In result, while the GNN is learning to reproduce a graph structure from the collection of samples, it is expected to learn how to automatically identify isomorphic graph structures (and possibly even graph sub-structures) and classify them into equivalence classes. This in consequences ensures that the learning is influenced by a strong inductive bias, as the learner prioritizes the similarity of structure over attributed values. By using only symmetric aggregate functions, the permutation invariance property is preserved. Some examples of good aggregate function candidates are element-wise summation, mean, maximum, softmax [15].

4-1-4 Connecting multiple GN-blocks into GNN Architectures

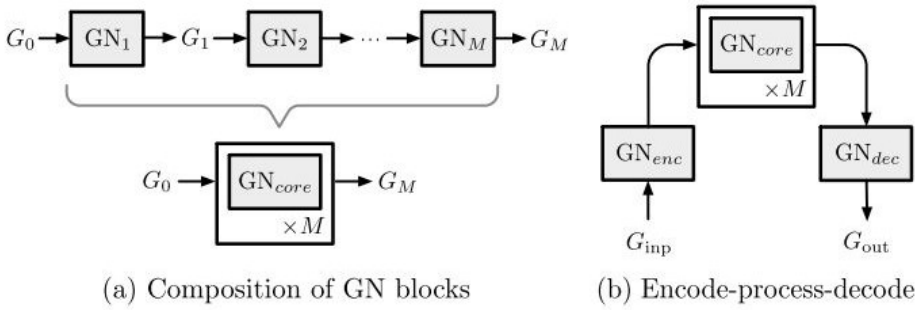


Figure 4-2: (a) GN-block series-connected to form the "core". (b) The encode-process-decode structure. GN_{enc} encodes an input graph, which is then processed by a GN_{core} and decoded by a third GN_{dec} into a desired task-specific output format. Source: [15]

An important advantage of designing GNNs from GN-blocks is the easiness of constructing interconnected multi-element architectures. It is possible, due to the "graph-to-graph" property of GN-blocks, that maintain the original graph input G_{in} structure throughout the update procedure. Output G_{out} is constructed from the same set of elements, with their attribute values updated in the update procedure discussed in the section above. In effect, even blocks that have different internal configurations can pass the graphs between each other. An arbitrary number of M blocks can be joined in a series connection. This results in a GN's update function composition that can be denoted as: $GN_1 \circ GN_2 \circ ... \circ GN_M$. The final output graph can be calculated as: $G_{out} = GN_M(... (GN_2(GN_1(G_{in}))))$. Such architecture is presented in fig. 4-2(a).

One important feature needs to be pointed out. By choosing whether the update functions ϕ in interconnected blocks share the same set of weights, one can tailor a GNN for a specific prediction tasks, closely resembling compound Deep Neural Network architectures. In case when the weights are not shared, such that: $GN_1 \neq GN_2 \neq ... \neq GN_M$, the graph passes through an evolving series of differently parametrized, differentiable functions (i.e. neural networks). This resembles a Convolutional Neural Network passing an image tensor through consecutive layers varying in composition. Alternatively, when the weights of GNs in series

connection are shared, $GN_1 = GN_2 = ... = GN_M$, the same local update function is applied to a graph M times in process called message passing. In result, the series connection can be collapsed to a single unit, denoted by GN_{core}. The configuration corresponds closely to an unrolled Recurrent Neural Network.

Encode-process-decode[43] model presented in fig. 4-2(b) uses a core block that is surrounded by two additional encoder and decoder elements. The input graph G_{inp} is transformed into a latent (graph) representation in an encoder block G_{enc}, which is then processed M times over GN_{core} and at the decoder G_{dec} the final graph output is computed. In effect, beside optimizing inner functions in the core to produce useful updates of the elements of the graph, one can also use the peripheral blocks to learn optimized latent feature representation at G_{enc}, or to produce the output graph in some task-specific format at the G_{dec} i.e. by constraining the output to produce only node attributes and compare with a node-level target. The combination of message passing GN_{core} (using shared set of weights) in an encode-process-decode architecture is of special importance to the presented case, because such a model is by default chosen for the KGCN.

4-2 Message Passing Neural Network

At the very core of the developed KGCN model, in the GN_{core} block, is an algorithm that governs an iterative message passing procedure, based on MPNN proposed by Gilmer et al. [37] generalizing from a number of previously synthesized examples, including [53, 56, 14]. The most important feature of the MPNN is that it splits a forward pass of the GN-block update into two parallel phases (in terms of discrete time step measures T). First, a *message passing* phase runs for T steps, to be followed by a *readout* phase executed once at the end of the cycle. This approach is used to emphasize the influence of interactions between the elements of the graph. With respect to the previously presented fully-connected GN-block with update

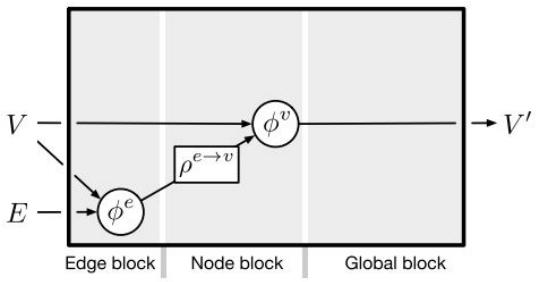


Figure 4-3: Modified GN-block used in KGCN Message Passing GN_{core}.

equations as in eq. (4-1), the GN-block used for MPNN proposed by Gilmer, presented here in fig. 4-3, has a few significant modifications, changed to fit a specific prediction task. Originally, the MPNN described in [37], was designed with a purpose of predicting targets at a node-level (a subset of 13 node attribute values), hence the node output V' was used for the loss calculation. It computes interactions on the edges based on the previous edges features, and on the features of the nodes sending into those edges. It then updates the nodes based on the incoming edges. Aggregate output vector of all nodes is produced by calculating the

readout function. Additionally, even though for some of the samples there was available data representing a global attribute, the output of the global was calculated, but not used as input into any of the update functions. Instead of an attribute MPNN uses a global node d_{master} connected to every other node of the graph as receiver. There is no equivalence of such node in the GN method. From the referenced paper it is slightly unclear what output it produced and how was it used. It can be assumed that in the MPNN the global attribute is absent and the final output is strictly at node-level. The implementation in KGCN sometimes diverges from the original MPNN, synthesizing solutions from different sources, i.a. [14]. The equations in eqs. (4-2) and (4-3) are presented in a form which describes, as closely as possible, the actual KGCN configuration.

4-2-1 Message Passing Phase

The equations governing the message passing procedure are governed by eq. (4-2). The message passing runs for T discrete time-steps and is defined in terms of message passing M_t and node update U_t functions. For better comparison, they are be represented in a consistent notation with a fully-connected GN-block update discussed in eq. (4-1).

$$\begin{aligned} \mathbf{m}_k^{t+1} &= M_t(\mathbf{e}_k, \mathbf{h}_{\mathbf{v},r_k}^t, \mathbf{h}_{\mathbf{v},s_k}^t) && := \phi^e \\ \mathbf{h}_{\mathbf{v}}^{t+1} &= U_t(\bar{\mathbf{m}}_{\mathbf{v}}^{t+1}, \mathbf{h}_{\mathbf{v}}^t) && := \phi^v \end{aligned} \qquad \bar{\mathbf{m}}_{\mathbf{v}}^{t+1} = \sum_{k=r_k=i} \mathbf{e'}_k \quad := \rho^{e\to v} \tag{4-2}$$

M_t, U_t are learned differentiable functions equivalent to ϕ^e and ϕ^v of eq. (4-1) without taking $\mathbf{u}$ as an input. The aggregate functions are implemented with element-wise summations. In this example, a shallow Multilayer Perceptron (MLP) neural network MLP with the Rectified Linear Unit (ReLU) activation and a output layer normalisation is implemented as each of the update functions. The transformation of messages is learned in order to pass useful updates around the graph. During message passing phase, at each time step t, hidden node state denoted as $h_{\mathbf{v}}^t$ - a temporary value encoded at each node in the graph - is updated based on messages $m_{\mathbf{v}}^{t+1}$ passed between sender and receiver nodes (denoted by indices s_k, r_k) through adjacent edges $\mathbf{e}_k$. An arbitrary graph updated through message passing is depicted in fig. 4-4. It is important to notice that the process is executed simultaneously between *all* elements of the graph such that they do not influence each other. After each iteration of M_t, $\bar{\mathbf{m}}_{\mathbf{v}}^{t+1}$ is aggregated globally across the nodes. Note one could also learn edge features in an MPNN by introducing hidden states for all edges in the graph. The output of the readout phase is a set of updated (but not aggregated) node attributes V' that are being passed to the readout function. Due to absence of the global attribute, there is no need for edge-to-global $\rho^{e\to u}$ aggregation, while the node-to-global is aggregated only after the readout phase.

4-2-2 Readout Phase

$$G^T = R(\{\mathbf{h}_{\mathbf{v}}^T \,|\, \mathbf{v} \in V'\}) \tag{4-3}$$

$$R := NN_R[\sum_{v\in V'} \mathbf{h}_{\mathbf{v}}^T] = f_{lin}(W\bar{\mathbf{h}}_v^T + b) \tag{4-4}$$

The readout phase is executed after T iterations of message passing, where a feature vector for the whole graph $G^T = G_{out}$ is calculated using a sum of final hidden states $\bar{\mathbf{h}}_v^T$. The

element-wise summation takes a role of $\rho^{v \to u}$, which is invariant to graph isomorphism. The differentiable learned readout function R proposed by [14] is a Linear Neural Network, which passes updated node attributes through a single hidden layer W with an added bias b. It is important to note that, in the current KGCN implementation, the readout is computed after each $T = 1$ step of message passing, so is the loss function. This is supposed to encourage the network to solve the problem in the least possible number of steps [61].

4-2-3 Message Passing in Encode-Process-Decode Architecture

The computations of the message passing take place in the GN_{core}. The input to the core is a concatenation of the encoder's output at time $t+1$ and the previous output of the core block at t. Furthermore, in the proposed architecture the readout function takes place in a separate G_{dec} block, that takes a graph representation with updated node values as the input. readout phase happens in the GN_{dec}. The GN_{enc} full functionality is yet to be explained along with KGCN solutions in section 4-5 as it serves as an interface between the graph output of Grakn and GraphNet tensor-based representation.

On importance of global attribute: On the final note, let's underline the meaning of absence of the global attribute $\mathbf{u}$, that compared to the fully-connected GN eq. (4-1), in MPNN is not used as an input of any update function. A global attribute broadcasts the information across all nodes and edges. When it is excluded from the update function argument, the information that a node or edge has access to after T steps of message passing is determined by the set of nodes and edges that are at most T-hops away from that of a node, where each "hop" is an edge-to-edge connection between a pair of nodes. The way the information travels in a graph is presented in fig. 4-4. If the number of steps is smaller than the largest distance between nodes in the schema, some nodes may never directly influence each. This can be interpreted as breaking down a complex graph computation into smaller subgraphs. It increases the influence of local interactions over the relational structure, however it may also decrease GNN's capability to capture long-range dependencies, which in turn has implications to the bias-variance trade-off of the training procedure.

4-3 Elements of an Ontology Design in Grakn

The content of this section will focus on description of elements of a knowledge graph in the Grakn-specific format. Concepts discussed here are implemented in the SPM schema design the next section section 4-4.

4-3-1 Grakn Knowledge Schema

Grakn [75] is the data-management engine designed specifically for explicitly structured representations of data. Grakn's query language - *Graql* - performs logical inference through deductive reasoning of data patterns and relationships [75]. Through Graql it is possible to design an enhanced entity-relationship type model schema [21] that describes interrelated *concepts* in a specific domain of knowledge, in that case: the SPM model. The inference is

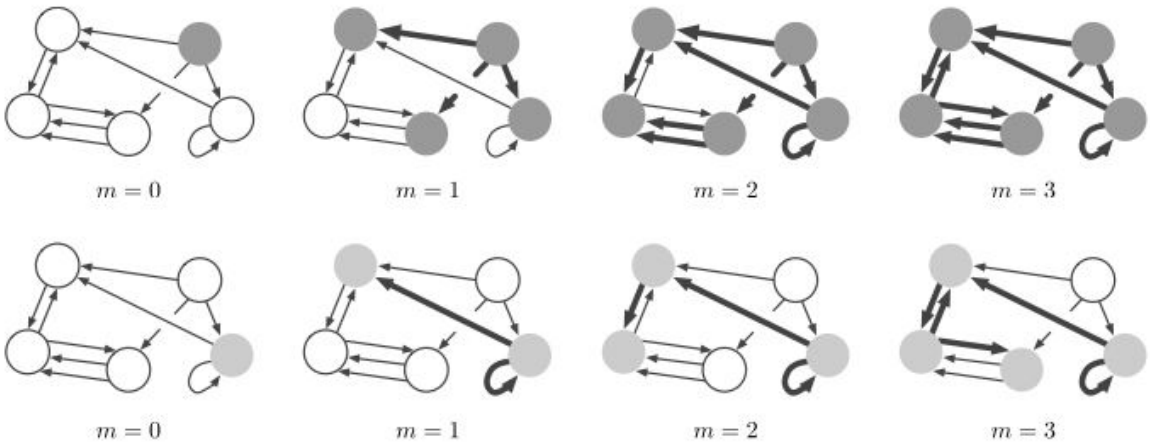


Figure 4-4: Example of message passing. Each row highlights the information that diffuses through the graph starting from a particular node. For comparison consider the two shaded nodes executing the message passing phase: the upper right (dark grey) and the lower right (light grey). The other shaded nodes and bolded edges indicate how far information from the original node can travel in T. Note that during the full message passing procedure, this propagation of information happens simultaneously for all nodes and edges in the graph (not just the two shown here). Source: [15]

performed through processing of entity and relation-type nodes network, as well as rule-based reasoning. The ontology is written into a separate `.gql` file that is loaded into a Grakn Core engine and by itself it does not yet include any data instances. They have to be migrated in the later process, effectively "loading" the designed knowledge model with useful examples. The data migration step is an important, yet very technical code-based process, so its description will be skipped in the main body of the report. The details on the data migration can be found in the Appendix B-1. For now it is sufficient to say that correctly loaded schema is the one that contains no duplicate entity or relation instances. Node's uniqueness is specified by a non-repeated combination of attributes that are attached to it.

First of all, the explanation of basic building blocks of a Grakn schema. There are three main types of concepts that serve slightly different functions in the inference mechanism. They are: entities, relations and attributes. In Grakn all three of them are represented as nodes in a graph. Nodes can be connected through labelled edges - roles - as presented on fig. 4-5. It is important to point out a technical difference in the assignment of attribute values between Grakn implementation and previously discussed GN theory. Real-valued attribute vectors of edges $\mathbf{e}_k$ and nodes $\mathbf{v}_i$ are not directly available from the Grakn graph, because *the attributes are nodes* too, each of them containing a single scalar value or a string. It will show up in the KGCN application, for which it imposes a requirement to first create an attribute embedding vector to express each entity and edge in the graph. The details of this step will be explained later in KGCN related section section 4-5, yet for now it is important to be aware of the difference between these two representations.

Entities are the main concepts in the knowledge domain. These are usually describing closely related features or co-dependencies of instances in the data. They provide means of classifying objects in the domain by relating input features, that in a standard ML model

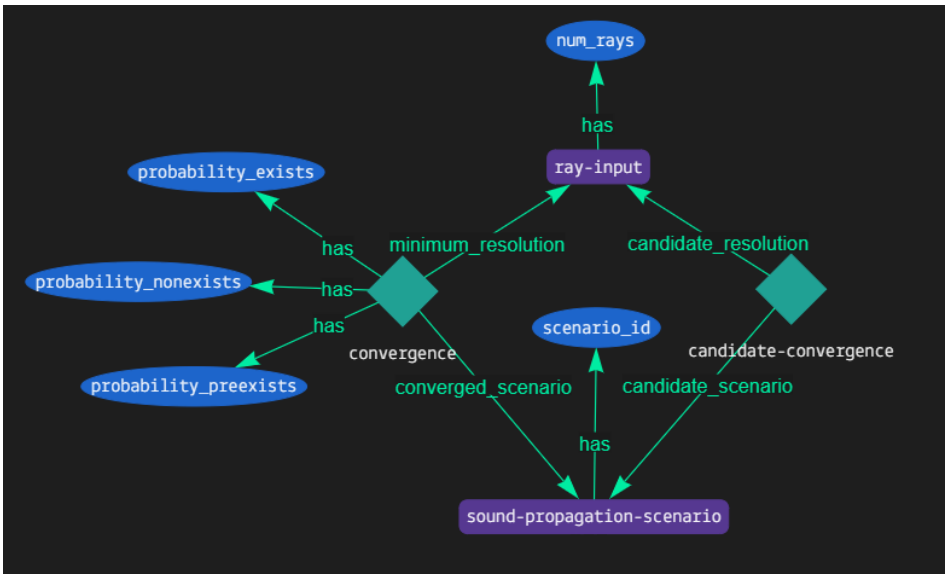


Figure 4-5: Simple Grakn schema model with all three basic concept types: entities marked in purple, relations in green and attributes in blue. The arrows denotes roles of "players" in each relation.

would be considered fully independent, or their true correlation would need to be learned.
Relations connect other concepts together by directional edges called **roles** that can be played in a relation. The relation node connects two or more role-players, where each of them has a unique role defined by a labelled, verbal description.
Attributes are used to express a single feature value. They can take following formats: numerical (64-bit integer or double-precision float), string, boolean (true or false) or date-time description `yyyy-mm-dd hh:mm:ss.fff`. Attributes can be assigned to entities, relations and even to other attributes (sub-attributes). It is possible to assign multiple attributes of the same to a single node. Each attribute is related to assigned node through special relation using a keyword `has`. This relation is always directed towards the "owner".
Rules are technically not concepts, but the resultant of automated reasoning engine over already existing concepts in the schema. However, they are an important addition to the latest Grakn Core releases and will play an important role in the design of the KGCN model. Rules provide a mean to insert an inferred relation between entities, which satisfy a range of pre-defined boolean conditions. All of the conditions need to be true for the rule-based relation to be inserted. In the presented example fig. 4-5, the `candidate-convergence` is a rule-based relation, that is inserted between entities `sound-propagation-scenario` assigned with a unique `sceario_id` attribute and `ray-input` that has an attribute `num_rays` representing one of 16 possible values for the target variable $\{500, ..., 15000\}$. The logics of rule inserts for the candidate relation can be formulated as:

Listing 4.1: Rule to create a candidate-convergence relation

```
1 WHEN {
2   $scn isa sound-propagation-scenario;
3   $ray isa ray-input;
4   IS NOT convergence(converged_scenario: $scn, minimum_resolution: $ray)
5   },
```

```
6 THEN {
7   MAKE candidate-convergence(candidate_scenario: $scn,
        candidate_resolution: $ray);
8   };
```

4-4 Proposed Ontology Design

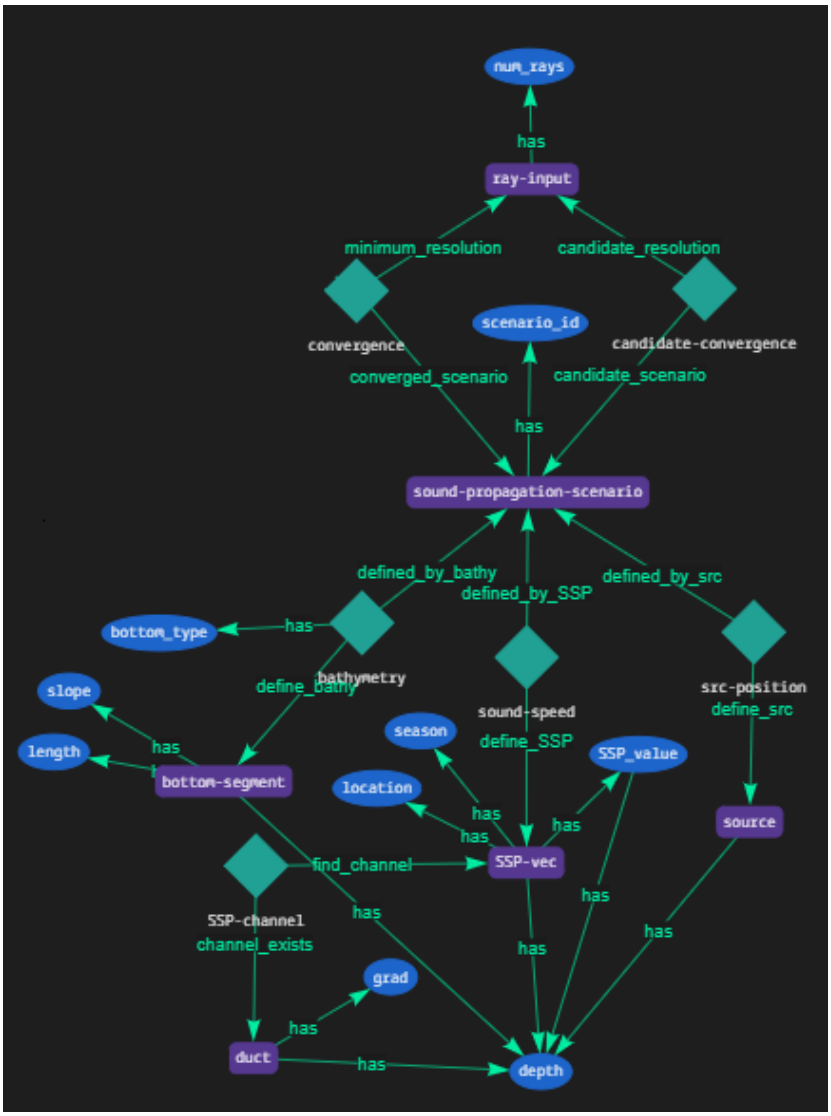


Figure 4-6: Knowledge graph schema developed for the BELLHOP model.

The schema design for the BELLHOP problem was an iterative process guided by a number of conceptual guidelines. These guidelines, presented below come from assumptions on the best structural representation of the BELLHOP model in a knowledge graph form, experience gained by TNO experts while working on Grakn-related projects and feedback provided by the Grakn team. In summary, the schema was developed with the following presumptions:

1. The schema should represent all BELLHOP inputs and additional features created for the XGBoost model (previously found best representation) in a relational structure using logical semantic statements.

2. The amount of added expert knowledge should be limited in the feature creation process, to test if the KGCN model is able to learn useful data representation. Nevertheless, the aim should be to include as much information as possible by interconnecting concepts through meaningful interfaces to represent their interactions.

3. Designed schema should be a stand-alone tool to analyse the collected database, such that it can be used for different projects. It is also to test the performance of database processing capabilities of Grakn.

4. The created graph should use the least possible amount of nodes that are highly interconnected. It shall not resemble a tree-structure as it was proven to be inefficient representation and also the knowledge graph shall not be mistaken for a taxonomy. It has to do with impaired message passing between nodes that are placed far away from each other. Also, querying performance is improved for graphs with less elements.

5. Example graphs, as the ones presented in figs. 4-7a and 4-7b, are obtained for each propagation by sending a query to Grakn memory. The obtained results for separate propagation scenarios should vary between each other structurally and not only by changing attribute values. It was observed that providing KGCN with distinct representations of example graphs reinforces the learning, as it leverages the influence of inductive bias from the relational structure. One should remember, that due to graph isomorphism, examples that are build on the same set of nodes, can be classified as equivalent. The model needs to learn how to generalise over a variety of graph structures, associated with some concept. After all, the combinatorial generalisation property is what really makes the GNN distinct from other DL models and allows for better exploration of capabilities of the graph data representation.

6. Preferably, the graph for each example should be obtained by the same query. The only variable changing in the query should be the scenario ID, equivalent to the row number in the collected database file. All other values should be filled in automatically. It greatly improves working with the graph database.

With the above considerations in mind and after numerous redesigns to improve the speed of the query, but also of data migration procedure, the final schema was obtained. It is presented in fig. 4-6. Full script of the schema can be found in the Appendix B listing B.1. Additionally, in table 4-1 the concepts are presented with their adjoint attributes, a short description of their function and finally the number of occurrences in the output example graph. The query that retrieves examples from the database is presented in the Appedix B listing B.2. The representation is highly compact, and it satisfies all of the above mentioned conditions. The explanation of schema design will be provided along with corresponding formulations used in the query to address specific parts of the graph.

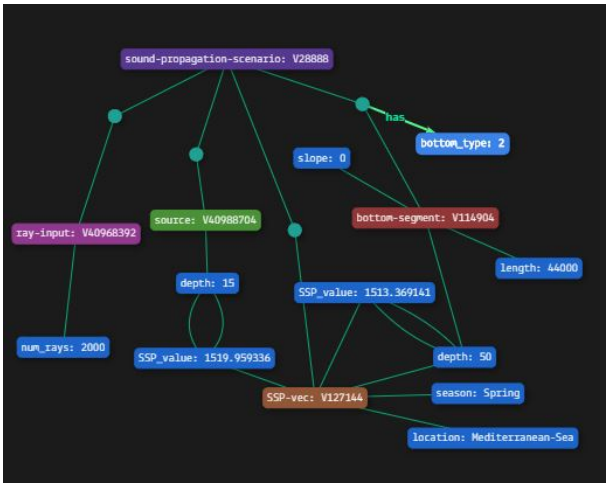
The presented schema in its basic form consists of 6 entities (marked in purple) and 6 relations, where the `candidate-convergence` is a relation inserted by a rule only in KGCN applications. Four basic relations connected directly to `sound-propagation-scenario` node describe the

Table 4-1: Schematic description of proposed concepts with their functionality and the number of occurrences in the graph.

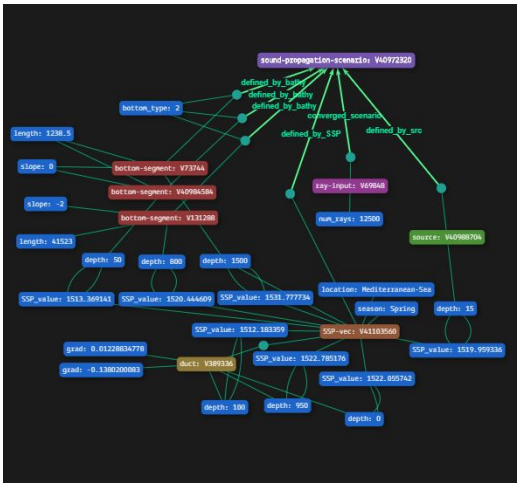
Concept Type	*has* Attribute	Function	Occurence
		Entity	
sound-propagation-scenario	scenario_id	propagation scen. instance	1
ray-input	num_rays	converged number of rays	1-16
bottom-segment	depth, slope, length	bottom boundary	1/3
source	depth	source position	1
SSP-vec	depth, SSP_value, location, season	SSP vector representation	1
duct	depth, grad	depth-boundaries and gradient of identified acoustic duct	0-1
		Relation	
convergence	probability (x3)	ground truth	1
bathymetry	bottom_type	bottom boundary influence	1/3
src-position	-	source location influence	1
sound-speed	-	SSP vector influence	1
SSP-channel	number_of_ducts	acoustic duct influence on sound propagation scenario	0/1
		Attribute	
SSP_value	depth	mapping sound speed values to certain depths	2-7

influence of: convergence, bathymetry, SSP vector and source position. Additional relation `SSP-channel` connects the `SSP-vec` node and potentially existing acoustic ducts that influence the sound propagation mode. These relations aim to categorize input features of the model into logical groups. In every graph obtained with a query listing B.2 the number of entities and relations may vary depending on the complexity of the example. To give an example: in scenarios with a flat bottom, there will be only a single bottom segment and a bathymetry relation node. However, in scenarios with declined or inclined bottom, it has been decided to represent the bottom boundary in three piecewise-linear segments. The beginning and end segment are flat and of equal lenghts (see fig. 2-8) and the middle segment is the inclined part. Each segment is attributed with a slope, length and depth value. The flat parts share the `slope` attribute equal to 0 deg/m and some `length` attribute, and have separate `depth` attributes equal to minimum and maximum depths in the water column. The sloped segment has slope attribute value -2 or 2 deg/m, some length, and depth value of the midpoint of that segment. In result for flat-bottom scenarios the bottom boundary is described by 1 node: `bottom-segment` and 3 attributes: `depth`, `slope`, `length`, while for the inclined case it adds up to 3 nodes and 7 distinct attributes. Next, to that the number of relations varies between 1 and 3. The increased representational complexity of the inclined or declined scenario goes in pair with the observed tendency for these to be much harder to solve, as proven by the end of Chapter 3. Similarly, if the duct is identified on a given SSP it can be a surface duct with a single (Sonic Layer) depth attribute and a mean gradient of the SSP function slope within its boundaries, or it can be a deep channel with 3 `depth`s marking its boundaries and the axis of the channel, and two `grad`ients associated with the average slope above and below the axis. Thanks to this variety, the structural difference in example graphs is created. In effect, the output can be as simple as the graph presented in fig. 4-7a or as complex as in fig. 4-7b.

These examples are the extrema of the scale and the are possible variations in between.



(a) Simple graph structure for a flat-bottom, no-duct scenario.



(b) Complex graph structure for a sloped-bottom scenario with a deep sound channel.

The most important concept group and at the same time, the most difficult feature to express, is the SSP vector representation. Based on the insight gain in the XGBoost experiments it has been decided to follow the approach that yielded best prediction results and compose SSP feature input distinctively from SSP values at the "critical depths" related to the other elements of the graph. So the SSP values of the `SSP-vec` are collected only for depths appearing in the other parts of the graph, i.e. source depth or min/max depth of the water column. In effect the `SSP-vec` is directly connected to a single `depth`, which represents its endpoint, but it is also indirectly connected to each other depth through its `SSP_value` attribute. This was achieved by first specifying `depth` to be an attribute of `SPP_value` attribute which automatically creates a vertex between these nodes. It resembles an injective 1-to-1 mapping between SSP values and depths, the functionality that is yet not implemented in Grakn. To fetch only these values (and not the whole vector) from the memory, a disjunctive or-statement is needed in the query (Lines 13 and 14 in listing B.2). It limits the output for `SSP-vec` node to SSP-values `$sspval` that interlinked with other entities in the scenario: `$dscr` source depth, `$dseg` bottom segment depth, `$ddct` duct boundary/axis depth. Otherwise, in the absence of this disjunctive statement the output would contain the whole range of sound speed - depth pairs, which are not necessarily meaningful and make the graph to become extensively large and hard to process. The use of this formulation has an additional advantage of creating multiple new links between branches of the graph to the SSP vector and its values, through the *shared depth attribute* vertex at a 2-hop distance. This is to express the correlation of the SSP vector with all other input features of the model. It also encloses the structure of the graph and should allow for better message passing, as the relative distance between concepts is decreased.

4-5 KGCN Model & Code Structure

Ultimately in this section the GraphNet framework [15], GNN theory, and the design of the Grakn schema come together in a complete KGCN model structure explanation. KGCN ap-

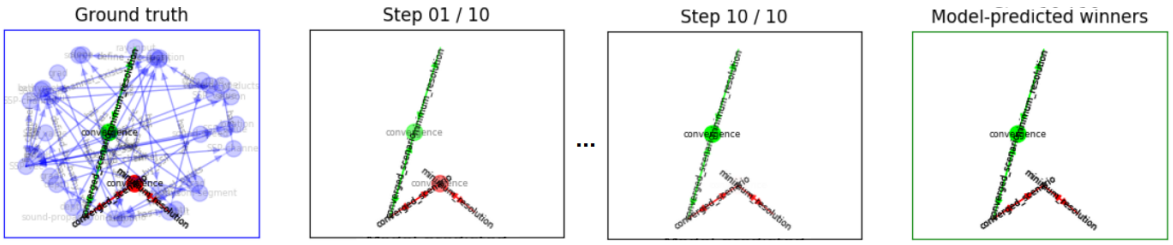


Figure 4-8: Depiction of Grakn learning to label correctly of a positive example *convergence* against negative example *candidate − convergence* in a binary classification task. The blue box shows the ground truth example, where: pre-existing (known) elements of the graph are shown in blue, relations and role-edges that should be predicted as true are shown in green, ones that should be predicted not to exist are shown in red. Black boxes show the progressive change in prediction as the model is trained. The opacity is used to reflect the probability assigned to graph elements exist/non-exist. Therefore, for good predictions we want to see no blue elements, and for the red elements to fade out as more messages are passed, the green elements becoming more certain. In the way training was defined, the model does not take into account the edges (roles) and tries to predict only convergence relation nodes. This justifies the presence of red relations in the final prediction (green box). All in all, the figure depicts an example graph that would be solved completely correctly (expected outcome). For a full dataset, there would be 15 red triples related to negative examples and one green triple for the true value.

proaches inference of a Knowledge Graph in a supervised learning setup. An example to illustrate the process, but in a binary classification task is presented in fig. 4-8. The goal is to predict whether there exists a `convergence` relation between the node `sound-propagation-scenario` (characterised by its unique key `scenario_id`) and node `ray-input` with a single `num_rays` attribute containing one of the 16 possible answers. The model is trained on a ground truth graph retrieved from the database by a specifically designed query to Grakn. They include all feature values related to a given scenario in a format dictated by the schema described in the previous chapter. They also contain a single correct `convergence` relation with the correct (ground truth) minimum number of rays and 15 other `candidate-convergence` relations, that serve as the negative samples for the supervised learning setup. Labels of these relations are obfuscated before they go into the learner to avoid information leakage. Effectively, they are all presented with the same label and the model should be able to predict which one of the 16 relations is the correct one. In this sense, the problem resembles One-vs-Rest multiclass classification problem, like the one solved previously with XGBoost.

Previously in section 4-1 it was explained how the core algorithm of GNN learning - message passing - transfers the information around graph nodes. In this section, KGCN is presented in a bigger picture, accounting for multiple formatting steps taking place in between retrieval of example graphs and the output of the learner. Arguably, the most important part of the workflow that was yet not explained is the Grakn-to-NetworkX-to-Tensor encoding and creating separate Neural Network embedders for each feature vector, that effectively take a place in G_{enc} block of the encode-decode-process. Merging the Graql language with TensorFlow application is possible thanks to another Python library: NetworkX [7] that allows for a string-based Graql output to be turned into a multi-dimensional Python dictionary object to represent all basic graph elements: nodes and edges together with the structure: directional edge-to-edge vertices and the adjacency matrix used for the overall graph structure. It facil-

itates easily separable dictionary classes for edges, nodes and attributes exclusively. Objects in these classes are the attribute values, stores in a structured manner with corresponding indices that can be iterated over. This is finally turned into a tensor representation for the input of the learner. The overall order of actions that can be visualised on a chart in fig. 4-9, below it will be explained by taking into account in which module of the code library is the action taking place. There are four main components that build the KGCN model on top of the previously created Knowledge Graph: Application, Pipeline, Learner and Core.

Application Found in *kgcn_spm.py*. This is the outermost module of KGCN code that contains schema-specific variables and query definitions. It is the only executable file that connects directly to the schema and data loaded in Grakn Core.

1. Find the Types and Roles present in the schema. Each attribute has to be specified as numerical or categorical variable and its range of values needs to be provided.

2. Fetch example graphs from the database saved in Grakn Core memory. This requires specifying queries that will retrieve Concepts from Grakn, based on the design schema. The queries are also used to create positive (`convergence`) and negative examples (`candidate-convergence`) for the supervised learning setup.

3. Perform prerequisite graph formatting and manipulation to transform the output query (queries) into a single output graph with full information It includes two important steps:

    - Defining and assigning labels to the nodes which are known to exits and those to be predicted by the learner. In the presented case the target of interest are nodes representing a `convergence` relation. Their edges `minimum_resolution` & `convereged_sceario` are not taken into account while evaluating prediction (i.e. in loss function calculation). This is illustrated in fig. 4-8.
    - Merging output of multiple queries into a single subgraph. Due to disjunctive patterns in the query ("or" statements) and multiple nodes of the same type present in the schema (i.e. 3 bottom segments), the output of Grakn consists of multiple output queries representing a whole graph. The difference between them is a single variable, because all possible combinations of the output variables were retrieved. A full example graph, like the one in fig. 4-7b, consists of concatenated results of all partial output queries.

4. Transform the previously merged example graph into NetworkX formatting.

5. Save retrieved examples in local memory of a computer to save time in the future runs. Depending on the size of the schema and the uploaded database, querying can be slow, so this step saves a lot of time. Before creating new graphs, the application checks if a graph with specific scenario ID already exists in the provided local directory.

6. Divide graphs into training & generalisation, and optionally validation split and pass it to the *pipeline.py*. Validation is performed only with a trained GNN model on unseen part of data.

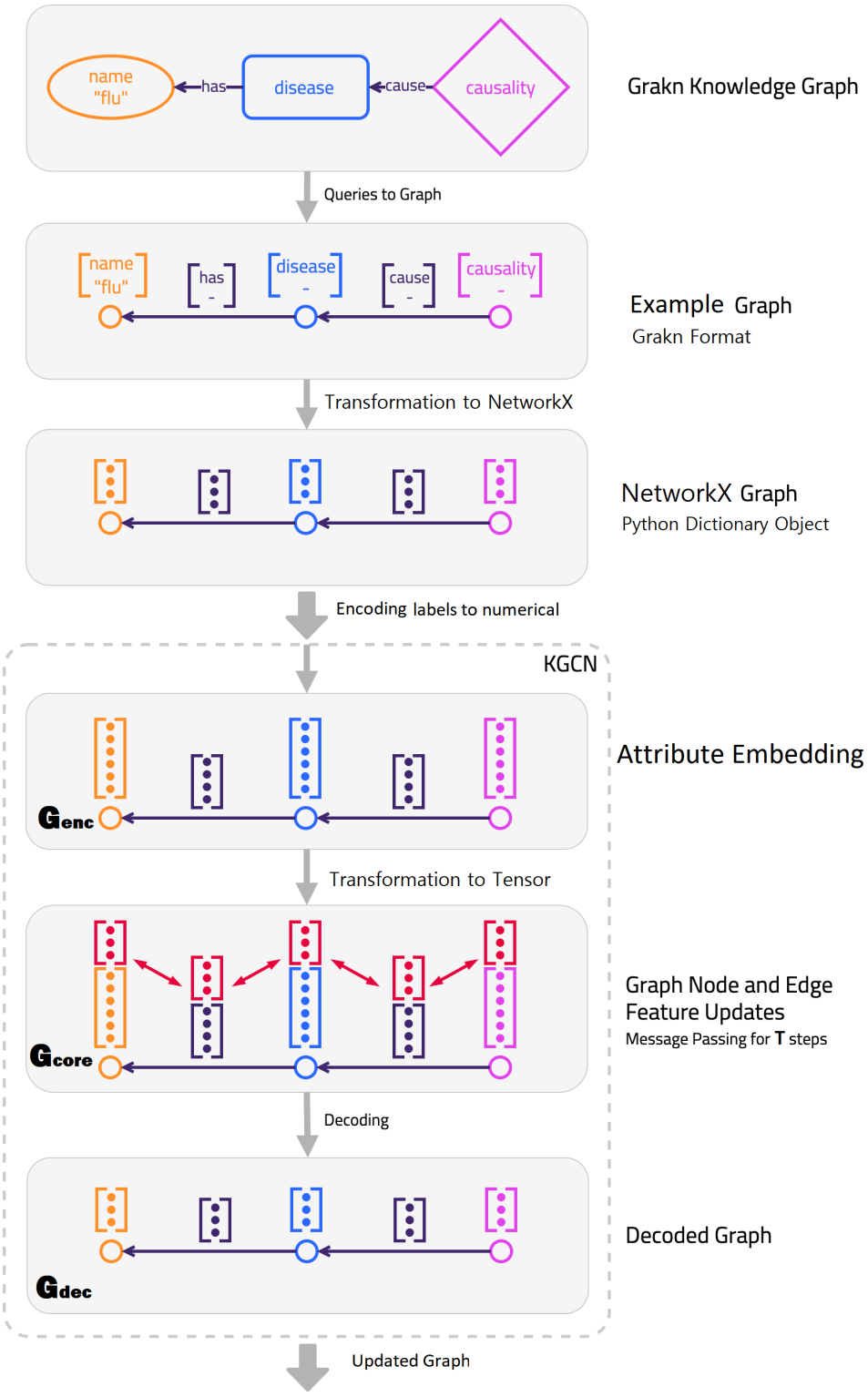


Figure 4-9: KGCN workflow presented on the arbitrary schema example. Source: Grakn-labs/kglib Github [39]

7. After the training is done, it is possible to write the predicted relations back to Grakn schema trough an insert transaction, therefore filling in values for the `probability_exists`, `probability_nonexists`, `probability_preexists` attributes of the predicted convergence relations.

KGCN Pipeline Found in *pipeline.py*.

1. Take in NetworkX graphs and perform value and label encoding, that consists of:
   - turning type labels into numerics
   - one-hot encoding for categorical features
   - normalising numerical features
   - preserving information about the ground truth label assigned to the elements of the graph that are known to exist
2. Take in transformed NetworkX graphs and create an Embedder Neural Networks for each feature of the model. Embedders are two separate modules found in G_{enc} encoder block of the encode-process-decode architecture:
   - *Thing Embedder* creates an embedder for each node: entities, relations and attributes; provided embedding dimensions for each node type, categorical or continuous attribute classification
   - *Role Embedder* creates an embedder for all roles (edges)

   The output of this step is a column vector of pre-defined length (latent representation) for each node and edge element of the graph. They can be optimised in the GNN learning process in aim to find the best feature representation. After the models were created, an initial embedding is performed. That transforms graph elements retrieved from the initial Python dictionary object to tensor-based representation that can be fed into the MPNN core learner. The entity and relation nodes that do not carry any values, are assigned an 'empty' embedder of matching output size, just to transform them into tensors, so their attributes can be updated in the message passing.
3. Call *learn.py* to run the learning process.
4. Create visualisations and plot learning curves.
5. Record the predictions made, and return them in graphs.

Learner Found in *learn.py*.

1. Performs the training loop.
2. Performs the validation loop.
3. Manages the loss function and optimiser.
4. Manages the TensorFlow session.

5. Prints results for the training and generalistaion datasets during training.

6. Uses the writer to save the training output and the model into a predefined, time-stamped folder.

KGCN Core Found in *core.py*. Defines the computation graph for a KGCN, including the initial embedding of values and the edge/node/graph feature update strategy during message passing. This is the core that depends upon Graph Nets [15]. Current implementation uses an outdated version of TensorFlow 1.4. This comes with some limitations regarding computational performance, that are discussed in section 4-7.

4-6 KGCN Model Tuning & Testing

The KGCN model is designed to solve a relation prediction problem, which is a classification task in a supervised learning setup. It should predict the existence of one of 16 possible `convergence` relations associated with the correct resolution (number of rays) needed to solve an example of propagation scenario. This relation should be labelled as true, when its softmax-normalized probability value `probability-exists` is larger than `probability-nonexists`. The opposite shall happen for the other negative candidate relations inserted by the conditional rule presented in listing 4.1. In the training loop the graphs are first split into training and validation/generalisation sets. The GNN processes these simultaneously, calculating loss and evaluation metrics for each set. Loss function compares the output graphs by comparing probability values assigned to convergence nodes with one-hot encoded label that has a value of 1 for the existing relation and 0 for the others. It does not take into account edges, or pre-existing elements of the graph. The loss function used in the model is a softmax cross-entropy function, the same as the one used for the XGBoost model eq. (3-15). Gradients of the loss function are computed with Adam Optimizer algorithm [52] and propagated back to the neural network in the feedback loop. Every n-epoch number of executed training iterations the feature vector outputted by the KGCN learner is used to calculate additional evaluation metrics in training and generalisation sets. It used more strict condition on the existence of each element by calculating the $argmax()$ on the softmax probabilities output of the learner. The argument with the maximum probability is assumed to be the existing one and compared to the ground-truth graph. This sometimes causes some divergence between the metrics and the loss function that calculates gradients based on the actual probability values of each relation. The accuracy of labelling is averaged for the whole chunk of training and validation data and output as: Ctr/Cge the percentage of correctly labelled nodes or Str/Sge percentage of fully matching graphs:

- Ctr/Cge - training/generalisation fraction of nodes/edges labelled correctly
- Str/Sge - training/generalisation fraction of full examples solved correctly

In that sense Str/Sge is more strict, as it requires the complete predicted graph to match with the ground truth example, so all the nodes, both positive and negative ones, need to be labelled correctly. Especially in the multi-class scenario it is easy to imagine that, while

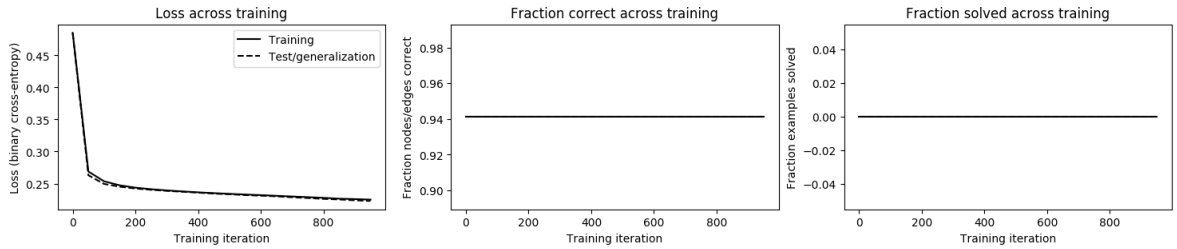


Figure 4-10: Results of the initial run with all classes and default KGCN setup. The classifier output labels all relations as non-existing.

the learner can usually easily exclude a few candidate relations, labelling of all 16 of these correctly is a much harder task.

The results of the training presented in fig. 4-10 on the whole dataset reveal a major drawback of the method: the loss function (left graph) tends to converge too quickly to zero as the classifier predicts each relation as non-existing. For each node, the probability value is initiated at $1/16 = 0.0625$ on each convergence relation node. The most probable answer is false, when 15 out 16 relations are false. This can also be confirmed by a constant value of the Ctr/Cge metrics (middle graph) at $15/16 = 0.9375$ and 0 correctly solved full graphs (right graph). A rapid decrease of the loss function and in effect too quick convergence of the gradients, what stops backpropagation from proper functioning and in effect weights of the neural networks used in the model are no longer updated. In conclusion, the model is not able to label the positive example correctly.

These results lead to a reveal of substantial flaws of the proposed approach. The intermediate conclusion is that obtaining a correct prediction in a multi-class classifier setup is impossible with the constructed GNN. However, a series of steps was undertaken to investigate if the model can learn the constructed graph representation at all and to exclude potential bugs in the code. The troubleshooting work resulted in a few important discoveries and model modifications that go beyond the original KGCN setup. In a series of tests it will be proven that the model is indeed functioning and learning to predict the correct label in a 2-class setup, however it performance drops drastically for increased number of classes. This is reinforced by analysing the training progress across elements of the KGCN model with use of TensorBoard environment and ensuring that the gradients converge to a stable value with time. The outcome of this work is presented below.

4-6-1 Model Modifications & Tuning Parameters

KGCN uses a large number of hyperparameters that could be tuned to improve the performance of the model, however only the ones that were found to be the most influential and may add an insight into functioning of the model will be discussed here.

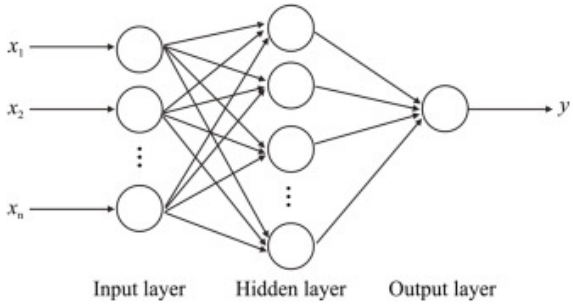
First of all, the biggest change with respect to the default KGCN model was introduction of the global attribute representing an abstract property of the overall graph. This value can be cast to all other elements in the graph without any direct edge connection. The modification was argued by the possibility of using too sparsely connected schema. Also in related literature, i.e. the original paper on MPNN [37], the experiments with implementation

of the global master node to allow long-range interactions, resulted in better scores on the predicted set. The importance of the global attribute was discussed by the end of section 4-2-3. Looking at the fig. 4-6, it could be argued that the convergence relation connecting `sound-propagation-scenario` and `ray-input` nodes creates a bottleneck between the lower part of the graph. A single link joins the relation that is about to be predicted with all the attribute values interlinked below the scenario node. It is possible, that this obstructs the message passing between the elements of the graph and the information does not transverse to the attribute with the number of rays. Next to that, in the MPNN-like GN-block the edges in the graph are never updated. It makes sense, when there is little information encoded in their attribute vectors, or the information describes a constant property, like in the case of the original implementation [37]. However in the case of the SPM schema, the information that travels through the edges is no less important than the nodes themselves. So it was decided to include the edge updates as well.

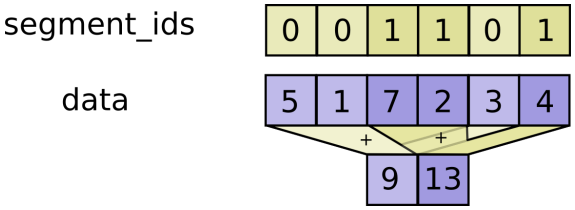
Proposed implementation for a GN-block internal functions: To facilitate the changes discussed above the default (MPNN-like) GN-block inner structure used in KGCN's GN_{core} was changed to a fully-connected one fig. 4-1b. This paragraph below gives the final definition of update and aggregate functions used in KGCN testing. The update equations of eq. (4-2) are extended with an additional neural network model for the global update and the element-wise summations are used as the aggregate functions. The global attribute can be used as an input of the message M_t and node U_t update functions. It was found to be beneficial to fix the global output size in the range 4-6, slightly larger than the size of node and edge output equal to 3.

$$\begin{aligned} \phi^e &:= NN_e[\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}] & \rho^{e\to v} &:= \sum_{k=r_k=i} \mathbf{e'}_k \\ \phi^v &:= NN_v[\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}] & \rho^{v\to u} &:= \sum_i \mathbf{v'}_i \\ \phi^u &:= NN_u[\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}] & \rho^{e\to u} &:= \sum_k \mathbf{e'}_k \end{aligned} \tag{4-5}$$

Because of introduced modifications the final proposed KGCN model uses equations of a fully-connected GN block, as presented in eq. (4-1), where each of the update functions ϕ is fitted with a separate, shallow (2-4 hidden layers) MLP Neural Network (presented here in fig. 4-11a) with a ReLU activation layer and output normalisation layer. The approach had been proven to be successful in a number of object-relation reasoning related research projects [43, 70, 67]. Different subscripts are used to stress the fact that these neural networks are completely independent from each other. No weights are shared between edge, node and global update functions. The inputs and outputs of the update functions are tensor-formatted graphs. The aggregate function input argument needs to match with the tensor format of the NN output. In the most simple way, it can be substituted by an *unsorted summation* calculated over selected axes of the tensor, called segments. Each segment contains a set of attribute values associated with an indexed node, edge or global element. The calculation is illustrated in fig. 4-11b. The function is invariant to the permutations of tensor values, as long as information about the indices of each element in the tensor-formatted graph are transformed along the way its value.



(a) Example of MultiLayer Perceptron (MLP) neural network substituted in eq. (4-5) for each of the update functions ϕ. Source: [44]



(b) Unsorted segment sum operation substituted in eq. (4-5) for each of the aggregate function ρ. The presented example is calculated on a rank-1 tensor with 2 distinct indices 0,1 provided in a vector segment_ids equal to the length of the tensor. The entries in segment_ids vector provide labels to tensor entries from which the segments are composed (equivalent to partitioning operation). Then, in each separate segment the elements are summarized.

Tuning parameters: After that, all the other parameters configuring the message passing phase were considered. They are controlling the number of steps of the message passing in the processing of the training and generalisation sets. As explained in the **??** the number of steps in the message passing phase can influence how far the information travels in the graph, especially in the absence of the global attribute. In the schema designed for the model fig. 4-6 the most distant node (`grad` attribute of the acoustic `duct` node) are at the distance of 8 steps away from the `num_rays` node. For that reason it was decided to run the tests with at least 10 iterations in the message passing and increase this number until the best result is achieved. Indeed, while the number of steps was set to low numbers i.e. 5 the training was extremely slow and generally not able to achieve the accuracy comparable with higher numbers of processing steps. It comes with cost of increased computation time. However, more steps may also lead to greater instability of the learner as the change in the final overall graph output, bigger loss and thus larger gradients. To prevent this, the gradient clipping was introduced. It is a technique that normally prevents the gradients in very deep networks from exploding by constraining the gradient norm below a certain threshold. Gradient clipping norms in the range of 5-10 yielded the best results.

Next, adjustments were made in MLP paramters used for the node, edge and global models. KGCN is an assembly of multiple Neural Networks that pass the information between each other. To simplify the design a lot them use the same structure and can be controlled by a single hyperparameter. It also allows for encode-process-decode structure to pass the transformed graph elements easily maintaining the correct shape of the output, as the GN-blocks re-use the definitions of the models for egde, node and global elements of the graph, even though NNs in each GN blocks are being trained independently. In the default implementation those models are MLPs with 2 hidden layers, 16 latent features, ReLu activation and output normalization layer on top. In the process of tuning the model it was found to be beneficial to increase the depth of used MLPs to 3-4 hidden layers, while maintaining the default latent size with 16 features.

Finally, as in every other neural network the parameter that has a great overall impact on the

progress and stability of the training is the learning rate. In the course of testing the KGCN it was discovered that setting a suitable learning rate can be very challenging. Too small learning rate causes the model to stagnate at too high loss value, while too large introduces instability that cannot be corrected by gradient clipping. The choice depends on the size and complexity of the dataset and should be complimented by adjusting the length of the training (number of iterations). It was found that the learning rate in range of $10^{-4} - 10^{-3}$ and 5000 iterations are sufficient to obtain a stable prediction score at acceptable training time. For the whole dataset an attempt was made to apply weights to the classes based on their frequency of occurrence in the dataset. However, this test did not result in an improvement of the prediction.

4-6-2 Biased 2-class Test

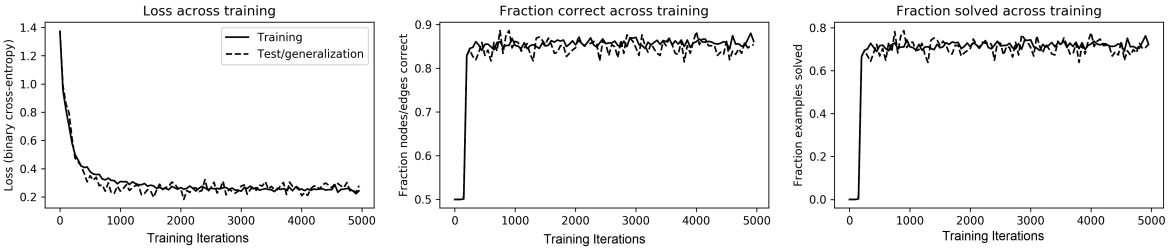


Figure 4-12: Results of the improved model training on a biased set with classes 500: 1000 samples, 2500: 10 samples. Added global attribute, increased MLP depth, tuned learning rate and gradient clipping.

To troubleshoot undesirable model behaviour, first it was decided to test if the KGCN model is capable of learning in a much simplified setup with a only 2 classes. To make the test even more explanatory and directed towards checking if the model is capable of simply producing the output in the desired graph format, the first test is executed on the heavily biased dataset of 1000 samples of 500 rays class and only 10 samples of 2500 rays class. The classifier trained in this way should learn to produce a naive output: by labelling that 500 rays convergence always is true, it still should be able to achieve a prediction score close of at least 99%. In the initial run with the default model setup two main issues were identified: loss function stagnating at a certain level (around 0.45) and not decreasing any further and instability of the training leading caused to gradients converging to quickly to zero. Also, the initial prediction score was in a range of 0.60-0.70 for the fractions of correctly labelled graph elements, which is relatively far from desirable threshold of 90% and above that a naive classifier should be able to achieve. The tuning was done with a primary goal to decrease the saturation level of the training loss function, while maintaining a stable end-result. The test was repeated with modifications discussed in the section above, yielding a satisfactory result. The modifications included changing some of the model parameters and introducing global feature. After the improvements the loss function decreased and the prediction score was able to reach 0.90 threshold in Ctr/Cge evaluation, however it stabilized in a slightly lower zone around 0.85. This suggests that the undertaken steps lead to measurable degree of improvement in training of the naive classifier. The set of found best parameters table 4-2 were used to train the model on the unbiased set, the results of which are presented in the next section.

Table 4-2: Best GNN parameters found in the 2-class biased test run. TR - training, GE - generalisation sets.

Tuning Parameters	Value	**GN-block Parameters**	Value
Learning Rate	1e-04	GN-block structure	Fully connected eq. (4-1)
Number of iterations	5000	Message Passing Steps (TR)	13
MLP depth	3	Message Passing Steps (GE)	13
MLP width	16	Edge/Node/Global output size	3/3/4
TR/GE split	80/20		
Gradient Clipping	7		

4-6-3 Unbiased 2-class Test

The next test was repeated in an unbiased 2-class setup. The input consisted of 1020 samples of the 500 rays class and 1020 samples of the 1500 rays (the whole available sample for that class). The model was trained for 10^3 iterations in the binary classification task with the best parameters found in the biased dataset. The results show that the model can be progressively learned towards achieving a approx. 0.73/0.7 Ctr/Cge accuracy score of the prediction and above 0.5 Str/Sge accuracy of solving complete graphs while maintaining stability. What is more, the loss and the evaluation metrics calculated in test and generalisation sets, converge to very similar values throughout the the training progress which suggests that the model does not have the tendency to overfit on the training set. Selected data at 0.8 tr/ge split creates a representative training sample. Finally, it can be observed that the gradient of the loss function gradually decreases by the end of the training, so hypothetically any further improvement in the accuracy of predictions would require much longer training or increasing the learning rate of the model, possibly even leading to overfitting at the training set. All of the above observations support the conclusion that the model is indeed capable of learning and solving a binary classification task on the BELLHOP database with certain, better-than-random accuracy.

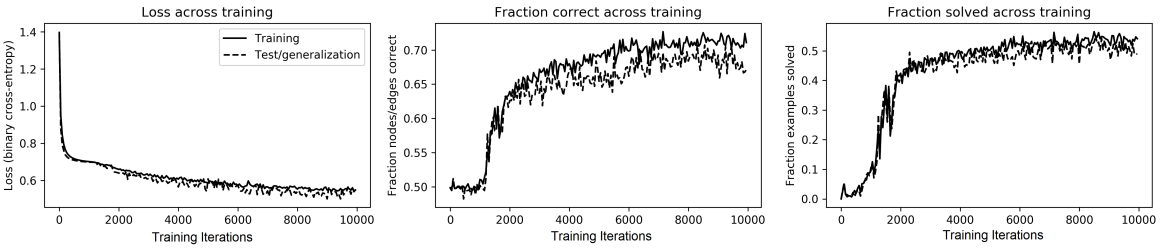


Figure 4-13: Results of a binary prediction task with the tuned model training on a equal distribution set with classes 500: 1020 samples, 1500: 1020 samples.

4-7 KGCN Model Revision

The outcome of KGCN model did not yield the expected results in the proposed setup classifier setup. As a proof of concept, it was proven that the model is capable of training a naive

classifier in purposely biased dataset of two classes, ergo to learn the graph representation correctly. It assures, that with high certainty the developed code if free of errors that could lead to wrong data retrieval, invalid graph transformations from Grakn into NetworkX format, data leakage, etc. The transformations into NetworkX dictionaries were selectively tested to ensure the correctness of the labelling and completeness of the information retrieved from the database. Some improvement of training effectiveness in terms of stability and decrease in the loss function was achieved. Nevertheless, even with the introduced improvements the model is still not capable of learning on the whole dataset. Obtaining a satisfactory prediction score required time-consuming hyperparameter tuning that did not always yield consistent results between the runs. Some issues could be mitigated but not fully removed. Compared to any other classifier trained in this setup, the Graph Network performs rather poorly. This leads to a conclusion that the method requires a throughout revision to result in a reliable graph neural network model. Unfortunately, due to time constraints of the project not all of the ideas could be tested. The section below aims to describe the most urgent of the identified problems and provides example solutions that could be adapted in further research work.

4-7-1 Identified Problems and Proposed Improvements

Problem 1: Solving classification instead of a regression task. Arguably the most problematic part of using the current KGCN model is the fact, that the model tries to reason out a single true relation against fifteen false ones, which even for a highly optimized algorithm like XGBoost was proven to be a difficult task. The unbalanced dataset with unrepresentative sample for the less populated classes (see chapter 3) adds to the complexity of the task. The reason for choosing this approach was mostly dictated by the only working example of the KGCN model presented in [40] to predict the existence of the relation between a patient and a disease. What makes the example different is that it operates on a highly synthetic and small database, and on a highly connected yet compact graph, that was further reinforced by introducing additional inference through Grakn rules not used in this schema (other than candidate convergence insert). The relation prediction is the only presented proof of concept for Grakn with KGCN application existing now. It served as a reference for testing of the developed Application, Pipeline and other modifications made to the code, also to gain an insight into functional GNN model. Even though the examples work as expected and the developed Application layer converts the example graphs obtained from the designed schema to NetworkX format with the correct labels, the model cannot be trained efficiently on the BELLHOP database. This suggests that the model architecture is not well-suited for the considered case. Modifications in the Core layer by replacing the GN_{core} block in encode-process-decode seemed to give a measurable improvement in the simplified setup, mostly due to the introduction of the global feature. However, the global feature property was not not used in the context that would exploit its full potential. It is an added abstract feature that does not represent any specific attribute in the schema. That gives a rise to a suggested solution presented below.

Solution: Redesign the model to perform a global attribute value prediction task, instead of a relation prediction. The global attribute shall represent the predicted number of rays. This would effectively make a KGCN regression, not a classification model. Such modification would require substantial changes in all layers of the KGCN model, but at the same time it could greatly simplify the model (and the prediction task). There are many benefits

coming from that solution. First, much of the troublesome steps with hard-coded labelling of graph elements as existing or "to infer" could be omitted leading to a simplified Application layer and NetworkX conversion step. Second advantage is an improved interpretability of the loss function and evaluation metrics. The loss could be simply calculated as RMSE of the predicted attribute value with the ground truth label, while the numerous evaluation metrics used to measure the accuracy of prediction of existence of graph elements would not be needed anymore. In that setup the whole graph is treated as a complete representation of a problem, that does not need to be verified in terms of existence of its elements, which is inherently done while comparing graph-to-graph output, even if the loss does not penalize the pre-existing graph elements. With all the above being said, it is debatable if such a model would be able to solve the regression task more efficiently than a standard ML algorithm, which can be seen in the lack of related research work in this area. The graph networks are usually used to solve tasks with graph-level (or at least node-level) targets, such as link or node existence prediction, graph completion, node-to-node interactions modelling, which is the main reason for using a such a complex multi-dimensional feature representation.

Problem 2: Non-optimized schema structure. The designed schema was written based on conceptual guidelines with respect to the composition of the graph, such that the examples vary between scenarios, etc. to enforce the structural inductive bias. However, to test if the schema works with KGCN, a specific Application module and a whole range of adjustments in the code first needs to be written and then tested in the overall setup. There are only a few intermediate steps one can take to test if the designed graph structured is a valid representation. In that way, it is nearly impossible to distinguish between the influence of the learner and the influence of the schema in the KGCN setup.
Solution: To optimize a schema a GNN model, according to the theory discussed in section 4-1-2, one would have to be able to measure an influence of relational inductive bias induced by graph structure, independent of data observations. The proposed method could be to start with the design of a test setup using graph representation directly in the NetworkX format. This would allow for creating nodes and edges with representable set of features from the real dataset, and manipulating them into different graph representations. A few different graph structures could be then tested in a simplified regression task to reveal a structure that would yield the best score. This would enable for a measuring the influence of the graph relational inductive bias before the whole dataset is fit onto it. Nevertheless, one has to keep in mind that optimising the schema is not a trivial task and could as well create a separate research project.

Problem 3: SSP feature representation The design of the ontology was driven by the need of representing BELLHOP database in Grakn format. It creates a major limitation with respect to the data format of attribute values that can be stored in Grakn Core memory. As of now, supported data types are limited to scalar values, bools and strings. This is what makes the SSP vector representation no less complicated, than in a standard single-column feature vector of a machine learning algorithm.
Solution: An idea to improve on the problematic representation of the SSP vector is to design a separate neural network to process this input separately and preferably learn to embed it into a lower-dimensional representation, while capturing the characteristics of the sound speed profile. As of now, in KGCN's G_{enc} block there are two separate embedder neural

networks. The Embedders are trained to find the optimal latent representation of assigned input elements. The SSP embedder would be different, only in a sense that, it would not separate the input based on any specific graph element type. Instead the dimensionality (or data type) of input is the dividing factor. What is important to realize, such solution would require feeding in the data from outside of the Grakn memory, which cannot save a vector as an attribute. It does however make a better use of full capability of GraphNet library, which as mentioned in section 4-1-3 was designed to work with arbitrary data formats. Possibly, to create a link between these two interfaces, one can design an SSP entity node with a categorical label, to serve as a handle (i.e. location and season, or geographical coordinates if a larger database is used) to an external database containing SSP data. The vector would have to be concatenated with the input NetworkX graph and altogether fed directly into the adapted encoder block. This would allow for maintaining clean graph-to-graph input/output relation.

Problem 4: Extensive consumption of GPU memory by graph placeholders. The KGCN model uses numerous hyperparameters that can be tuned to improve the results. Unfortunately the training, especially for a larger dataset (4,000 graphs and above in the case of designed schema), requires large computing resources, so grid-search based training methods could not be used. One of the main drawbacks of the current implementation is the lack of a batching mechanism for the graph input feed. In effect, the model first creates *graph placeholders* which are used for memory allocation for graph-based based tensor operations in TensorFlow . These placeholder structure can grow very large, as it is affected not only by the size of graphs, but also the number of training iterations and the number of parameters used in the neural networks. In fact, creating the placeholders for the whole set of 9031 graphs with the best found parameters: increased MLP depth, added global model, 13 message passing steps, for 5000 iterations takes approx. 23GB of GPU's memory. This leads to memory overflow issues, the training cannot be started.
Solution: Implement a batching mechanism for graph placeholders, which would require modifications in the Pipeline, Learner and possibly some secondary modules of the library responsible for the feed dictionaries generation. That would adapt the model to work with real, large databases and not only on synthetic examples. Alternatively, with the newest release of GraphNet library [61] the model could be converted to TensorFlow 2, which does not use placeholder data structures any more. As an added value, with TF2 the training can be performed on high performance TPUs (Tensor Processing Unit) available at Google Colab that come with 64GB of high-bandwidth memory. The solution is given with a disclaimer that the current version of KGCN does not support TensorFlow 2 by default, which can lead to package dependency issues.

Chapter 5

Conclusions

5-1 Research Summary

The goal of the research was predicting the optimal resolution of the acoustic Sonar Performance Model (SPM) from the data generated with BELLHOP algorithm. To solve this task two machine learning models were developed. First, to estimate the possible achievable accuracy of prediction on the collected database, an XGBoost model was developed. The choice of this algorithm was argued by its well-known high performance on most of the classification and regression task and robustness with respect to the feature selection and data distribution properties. Then, the main method of interest was investigated, resulting in the design of a Graph Neural Network classifier Knowledge Graph Convolutional Network (KGCN) to solve a relation prediction task in a supervised learning setup. The results obtained with the XGBoost has proven that the inference of the BELLHOP database is possible with a certain, satisfactory accuracy (nearly 80% F1-macro score). However, the proposed KGCN model turned out to be hardly applicable in the multi-class classification setup. It was possible to present a proof of concept in a simplified binary classification setup, that confirmed proper functioning of the developed model on a less complicated prediction task. The conclusion drawn from testing of the GNN model is that to solve the problem with better accuracy, the model should be redesigned for an attribute prediction rather than a relation prediction task.

A number of intermediate steps were taken in the process to reinforce the understanding of the BELLHOP problem with respect to data-based applications. First, the collected database was analysed in terms of data type, distribution and statistical correlations. The outcome presented that data collected in a large number of BELLHOP simulation runs poses range of adverse characteristics making it difficult to process with machine learning algorithms. Due to the fact that inputs to the simulations were discretized on a fixed, non-equally spaced grid, and so was the range of possible resolutions (expressed in the number of rays used in the geometric acoustic model) there was a limited variety of captured propagation modes. Also, the discretized resolution that in the reverse process becomes a target variable, creates a highly imbalanced set. Ray numbers used in simulation input vary between 500 and 15000 and the spacing in between consecutive values is not unequal. The characteristics of the

problem suggested that it can be better solved with a classification algorithm by treating the ray numbers as discrete labels, rather than a continuous value to be inferred by a nonlinear regression model. This was confirmed in a series of tests with XGBoost, yet failed to be true for more complicated GNN model.

Continuing, one of the major obstacles of working with this database was representing the complete BELLHOP input data in a machine learning input feature vector. The original simulation uses a very limited number of numerical input features, but it also uses the information about the Sound Speed Profile function that acts as a refractive boundary for the geometrical representation of the sound waves (ray tracking method). Effectively, the gradient of the SSP is the crucial factor that affects the propagation of the sound in the water column. The core of computation algorithm behind BELLHOP is calculating consecutive reflections and refractions of the rays passing through the SSP curve in a discretized underwater environment. Finding a good representation for this non-scalar input was very important to capture the information needed by the ML/DL model in solving the inference problem. To achieve this a number of routines were written for acoustic ducts identification from the SSP curve. XGBoost was used to benchmark a range of proposed representations, and in effect resulted in enriched rendering of SSP input that added measurable improvement of the prediction score. The results of this phase were propagated to the design of a Knowledge Graph schema for the GNN.

Regarding the progress in the Graph Neural Network section, obtaining a functioning model required a multi-stage design process. First, the ontology (or schema) of the knowledge graph needed to be proposed. The schema is a conceptual representation of the problem. It includes all model parameters grouped into logical categories with use of entities (nodes) connected by relations (edges) into a rich graph representation. The schema was developed in Grakn software that became a basis for the Deep Learning implementation. After multiple iterations, a compact and logically interconnected graph was created, that can be used as a stand-alone tool to work with BELLHOP database. Then a data-migration procedure was invented to load the schema with the data instances from BELLHOP output and SSP data. It is worth mentioning that written functions are capable of fully automated feature engineering to the desired format, that is consistent with the schema, but also with previously tested XGBoost models. The method used for the design of the GNN was based on the GraphNet [15] library by DeepMind. Their approach of composing the GNN architecture from GN-blocks (much like a Deep Neural Networks is composed of different NN modules) turned out to be well suited for creating a model to predict a convergence relation in a binary classification setup. The designed model works until certain problem complexity, although there is a range of improvements that still need to be introduced to make it applicable to work with a complete dataset.

In conclusion, the presented research proves that it is possible to perform accurate inference of the relations between features of the sonar acoustic model with standard algorithm. It is also possible to design a graph representation of this knowledge domain and compose a functional Graph Neural Network capable of the same task, however in the way it was designed it is not an effective solution.

5-2 Contribution

The contribution of presented research is two-fold. First of all, it was presented that data-based solutions may be a feasible alternative of predicting the required sonar system parameters in a simulated ocean environment based strictly on the collected data from BELLHOP simulations. It is a largely innovative approach that is not widely explored by the Ocean Acoustics experts. Machine Learning models can provide a much faster estimation by excluding the need of performing multiple time-consuming simulations in order to calculate a convergent, reliable solution. In this form they could be used in the applications that required quick approximations of the system performance, i.e. in online calculations on naval and submarine units. Alternatively, it could be used as a design tool for the sonar to suggest the performance requirements of the system. It would be interesting to adapt the method to work with the real-life data or in more complex simulated environments. The considered case used a largely simplified scenarios of sound propagation cases, that do not exploit fully the potential of the SPM model.

Secondly, from the perspective of data science, the developed deep learning model is a first ever designed Graph Neural Network to perform a prediction of the parameters of the sonar system. Due to complexity of the underwater sound propagation phenomenon, it can be beneficial to represent the knowledge domain in a form of ontology. This may allow for better understanding of interactions between the features in the SPM and lead to generalisation over its components beyond human understanding. In turn, this could be used to pursue further developments in this area to design machine-learning reinforced "smart" Sonar Performance Models capable of expressing complex domain-specific phenomena derived from the analysis of acoustic data.

5-3 Recommendations

The final section of this report is devoted to proposing further recommendations that could be used in the development of data-based models for applications in underwater ocean acoustics. The aim is to provide a comprehensive advise on the identified issues that were experienced throughout the research. For the detailed, model-specific guidelines please refer to sections 3-5-1 and 4-7 that give a better insight into a range of possible modifications to the XGBoost model and the KGCN mode at the ends of corresponding chapters.

- If the problem were to be solved with better accuracy of prediction, the data would need to be generated at the increased range and resolution of the input features. Among the others, the most important would be to increase the resolution, hence to reduce in size and possibly equalize spacings on the grid of the target variable. This shall allow for treating the dependent variable as continuous and training a regression model with better accuracy. Regression problems being fundamentally easier to solve than classification ones, could yield a better result in terms of prediction. An XGBoost Classifier achieved a better prediction score then a Regressor mostly due to sparse distribution of the target variable. Moreover, it was proven in the case of the GNN model that a multi-class classification is not a suitable representation for this model leading to corrupted training and in effect lack of correct predictions while working with the whole dataset.

- Another suggestion is to collect a more varied sample of sound propagation scenarios, Database generated of this research consisted in large of easy-to-solve propagation modes, i.e. flat and long water channels or deep water propagation where the surface ducts and acoustic ducts are trapping most of the rays at any resolution. This heavily reduces the type of phenomena that were registered. Also it would be highly beneficial to minimise the disproportion between "easy" scenarios (500-1500 rays resolution) and the more stable/difficult ones requiring higher resolutions. It was observed that especially high number of rays around 15000 are often related to the emergence of shadowing effect. It can be partially observed in fig. 2-7a, however the effect would be more severe in the absence of the trapping acoustic duct. This happens especially often for declined bathymetry cases, when the depth difference between the beginning and the end of the channel was large: i.e. 50m at the source position and 1500m at the receiver. In these cases the ray propagation starts in a heavily condensed shallow water and disperses rapidly at the corner of the bottom boundary where the decline starts. This gives a rise to a very chaotic propagation behaviour, that if it is not trapped in a surface duct, may result in very different distribution of the rays at the receiver. This is what causes large differences in Transmission Loss convergence calculations and in effect drives BELLHOP to run consecutive iterations until it reaches the end of allowable resolution range. However little information was captured in the mid-range resolution due to small amount of samples.

- Regarding the previous recommendation, it could also be beneficial to use real-life data instead of simulated data or increase the number of parameters that are changing between propagation scenarios of the simulated model. It has been observed many times that in the way the problem is posed now, it suffers from low dimensionality of the feature space which makes it very difficult to come up with accurate predictions without severe feature engineering. Lack of multiple meaningful features is also making it difficult to come up with a representative knowledge graph simply due to lack of concepts to be included in the domain.

Appendix A

XGBoost

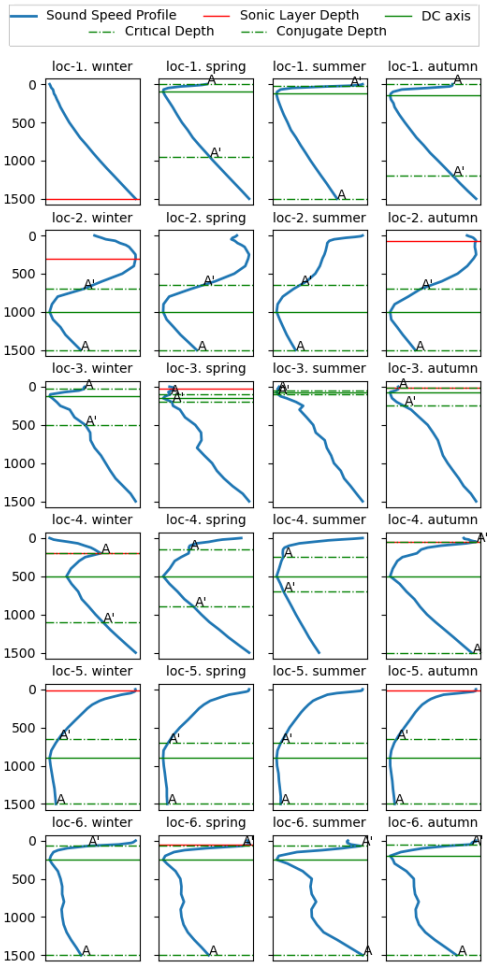


Figure A-1: Visualised output of the SSP duct identification algorithm.

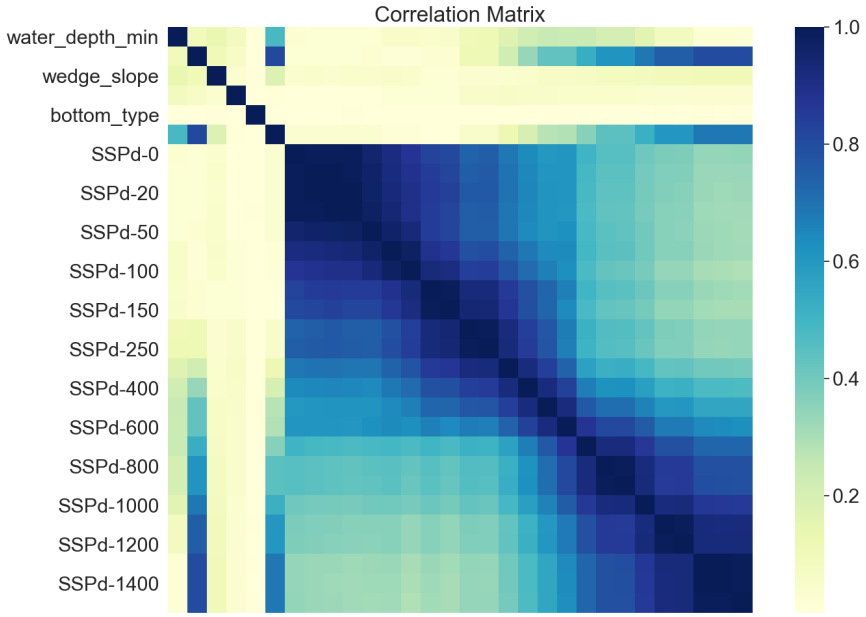


Figure A-2: Correlation matrix for vector representation of the SSP input.

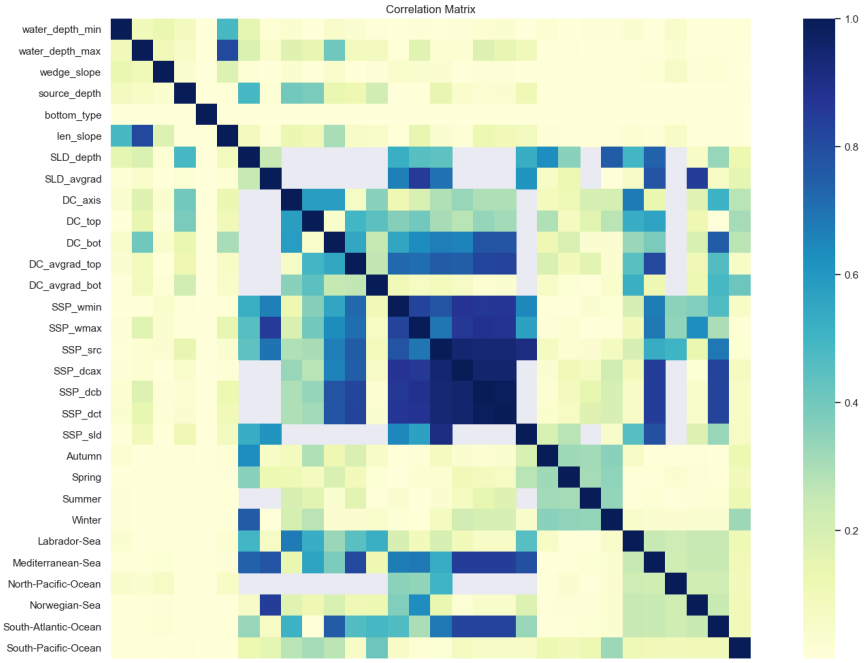


Figure A-3: Correlation matrix for the full SSP-identification representation.

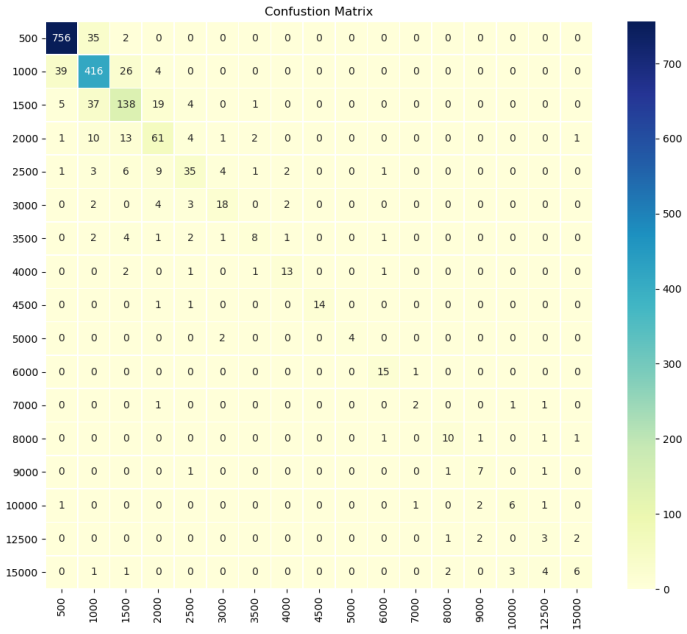


Figure A-4: XGBClassifier: Confusion Matrix

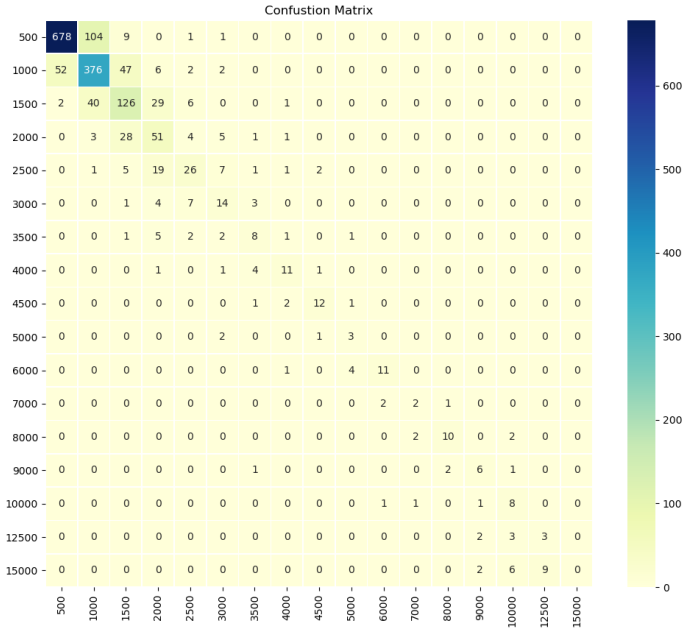


Figure A-5: XGBRegressor: Confusion Matrix

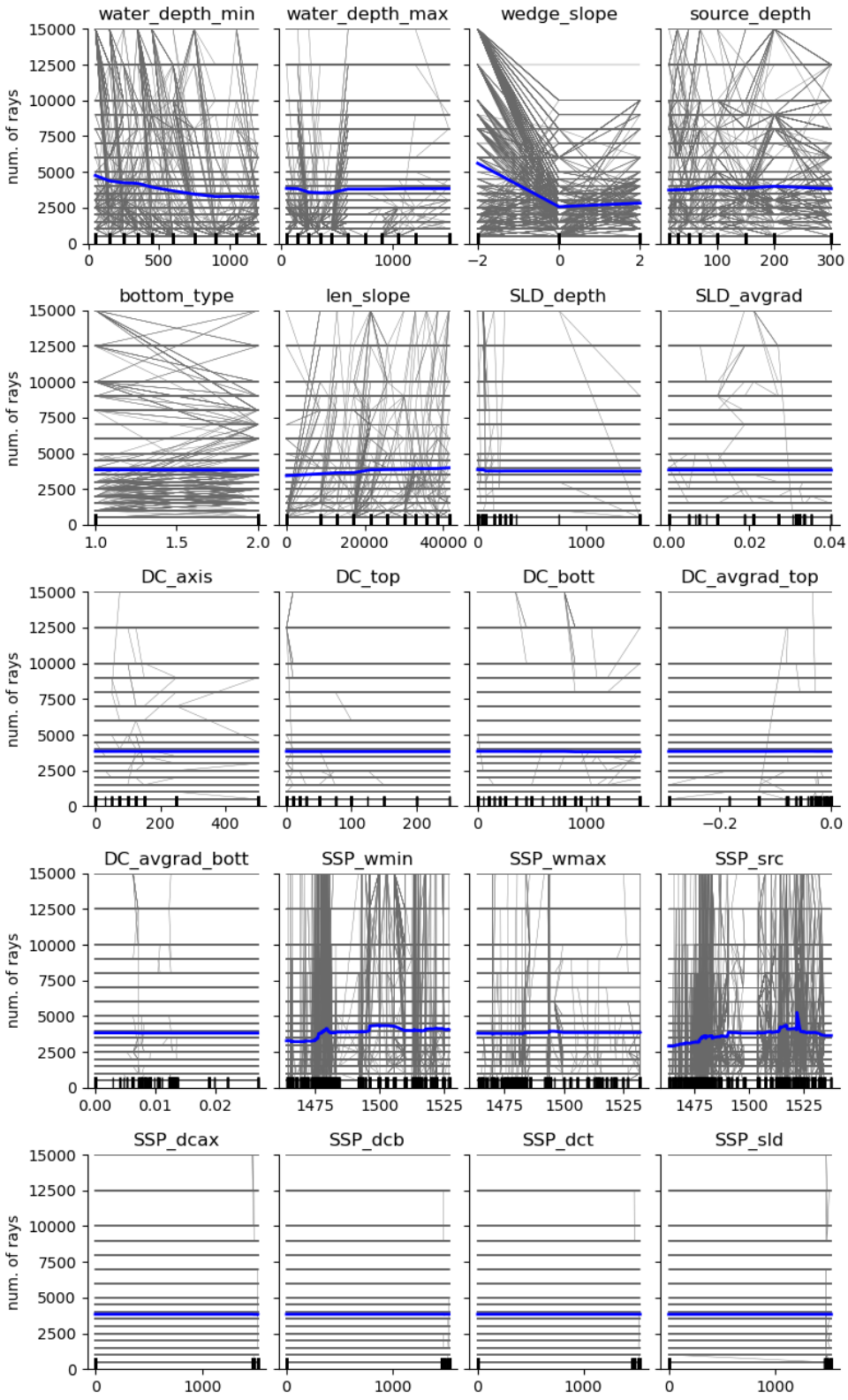
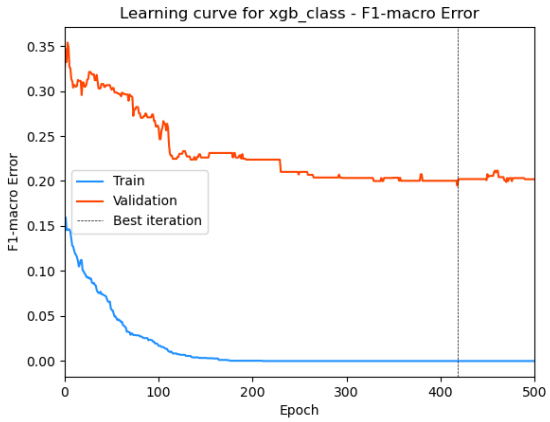


Figure A-6: ICE Plot

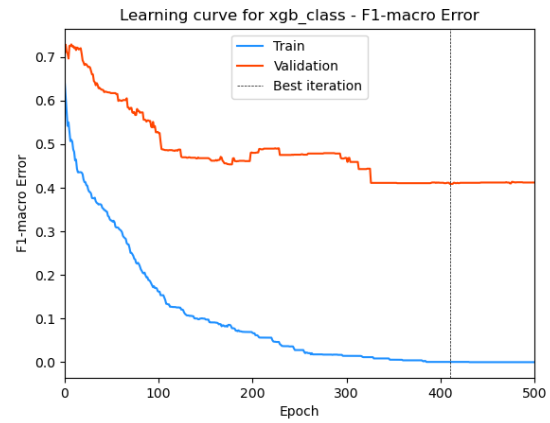


Class	PR	RC	F1	Sup
500	0.96	0.99	0.98	314
1000	0.92	0.88	0.90	159
1500	0.78	0.75	0.76	75
2000	0.64	0.54	0.58	26
2500	0.78	0.93	0.85	15
Macro Avg.	**0.81**	**0.82**	**0.81**	589
Accuracy	-	-	**0.91**	589

Figure A-7 & Table A-1: Slope 0 subset: learning curve and prediction report.

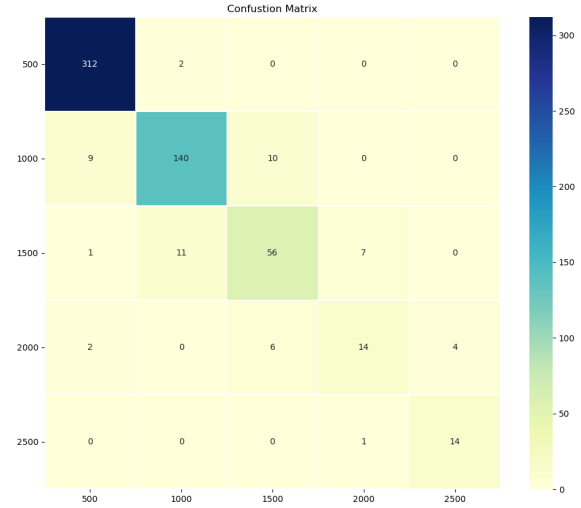
Class	PR	RC	F1	Sup
500	0.94	0.94	0.94	202
1000	0.86	0.94	0.90	214
1500	0.79	0.72	0.75	89
2000	0.81	0.73	0.77	48
2500	0.89	0.89	0.89	36
3000	1.00	0.77	0.87	22
3500	0.93	1.00	0.96	13
4000	1.00	0.75	0.86	12
4500	1.00	1.00	1.00	12
6000	0.72	0.81	0.76	16
8000	1.00	0.89	0.94	9
10000	1.00	1.00	1.00	9
Macro Avg.	**0.91**	**0.87**	**0.89**	682
Accuracy	-	-	**0.88**	682

Figure A-8 & Table A-2: Slope 2 deg/m subset: learning curve and prediction report.

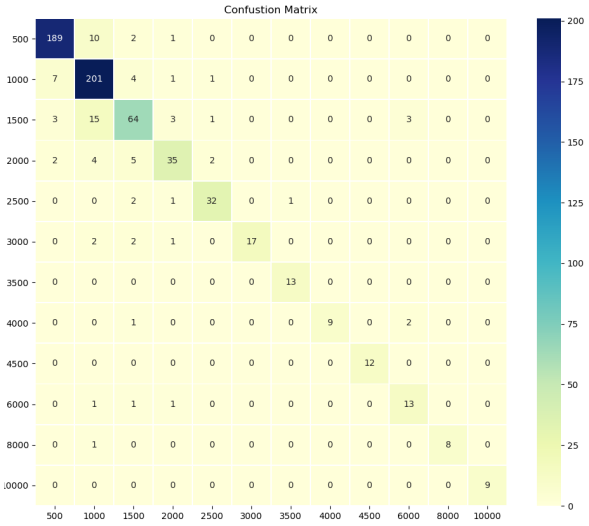


Class	PR	RC	F1	Sup
500	0.91	0.92	0.91	276
1000	0.68	0.71	0.69	113
1500	0.68	0.66	0.67	41
2000	0.80	0.67	0.73	18
2500	0.60	0.55	0.57	11
4000	0.75	0.60	0.67	5
8000	0.33	0.14	0.20	7
9000	0.67	0.57	0.62	7
12500	0.50	0.38	0.43	8
15000	0.77	1.00	0.87	17
Macro Avg.	**0.67**	**0.62**	**0.64**	503
Accuracy	-	-	**0.81**	503

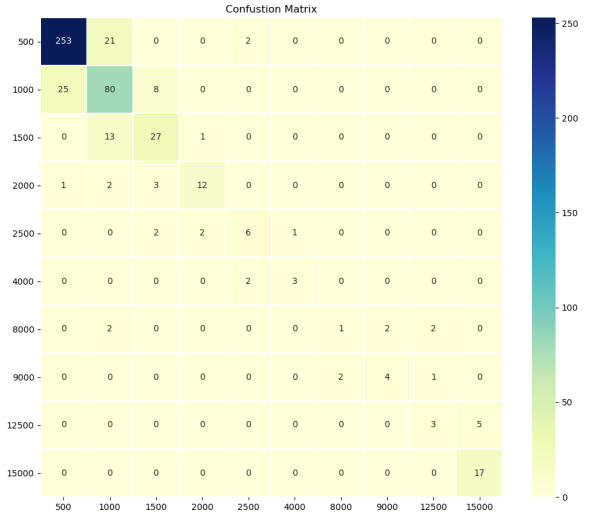
Figure A-9 & Table A-3: Slope -2 deg/m subset: learning curve and prediction report.



(a) Slope 0



(b) Slope 2 deg/m



(c) Slope -2 deg/m

Figure A-10: Confusion Matrices ofthe sub models split on slope.

Appendix B

Graph Neural Networks

B-1 Data Migration Procedure

The data migration procedure includes all steps that need to be taken to transform raw BELLHOP output into formatted feature representation and send it into Grakn Core memory through insert queries. It is an important process, that needs to be executed with special attention paid to the way that Grakn forms unique IDs for newly uploaded data instances. The outline of the Python script responsible for the migration will be briefly explained here. It was designed with an assumption to be easily reused for data in a comparable format, thus making it easily scalable for future applications with Grakn. The schematic representation of the algorithm is presented in fig. B-1.

There are three important considerations to keep in mind while migrating data into Grakn:

1. Each entity/relation instance has to be created by a query sent in a separate transaction. Grakn assigns a single unique ID per transaction, hence uploading a few nodes at once will results in invalid descriptor. The ID will be shared between all instances in the uploaded sub-graph.

2. One should avoid creating multiple non-unique instances of the same entity or relation. Concepts should be reused between the examples that yield the same results. That avoids retrieving duplicate records through match query, saves the memory and improves the overall performance. To illustrate this, for the presented example, in the memory of Grakn, there are 9031 `sound-propagation-scenario` entities, each with a unique key - `scenario_id`, but only 16 `ray-input` nodes each attributed with a different number of rays, or only 10 `bottom-segment`s that are characterised by unique set of depth and length attributes.

3. Entities need to be created before relations. That is due to mechanics of the relation insert (discussed in the section above) that first needs to match already existing nodes before creating a link between them. Hence the logical order of migration is important.

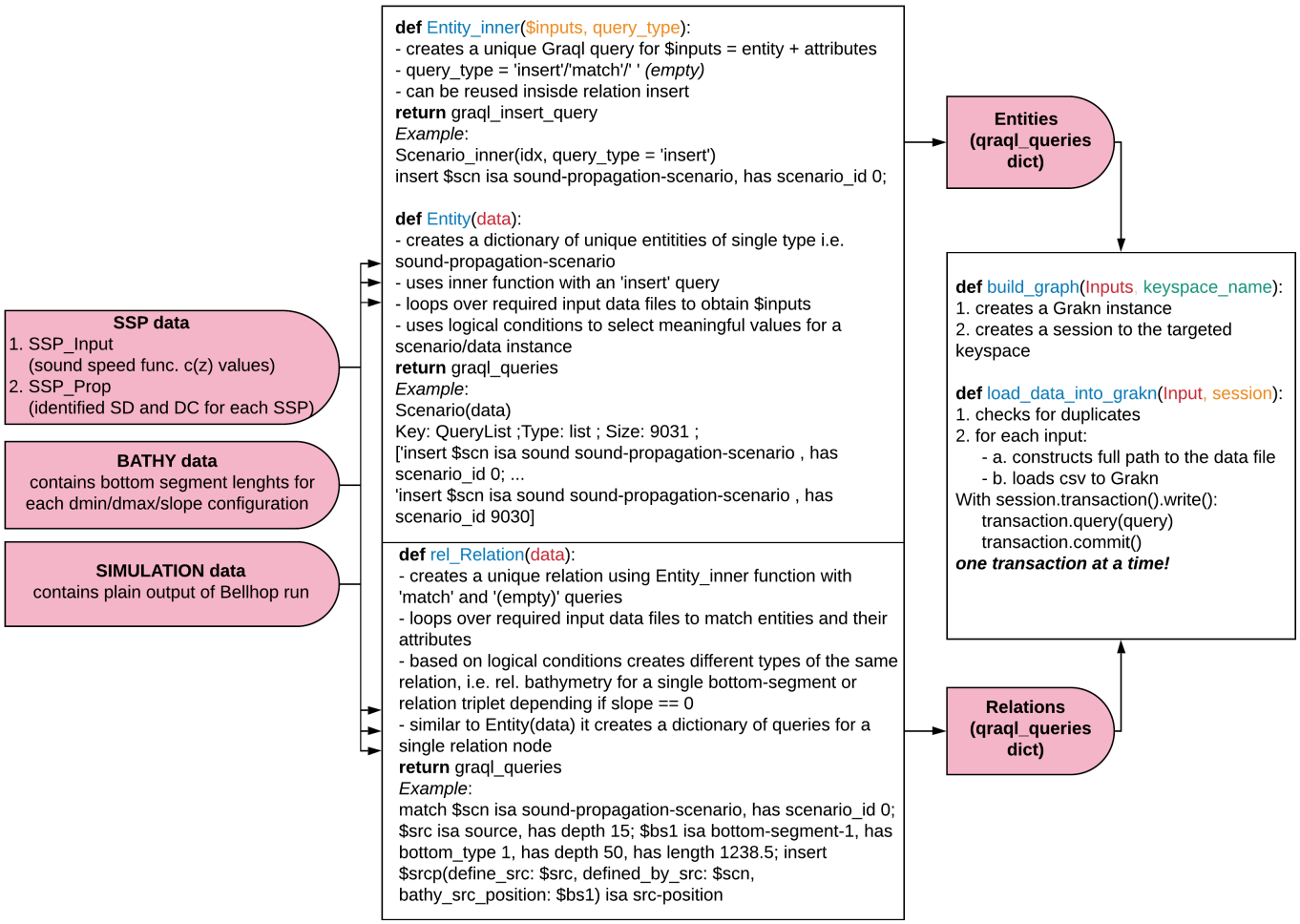


Figure B-1: Schematic code-block representation for the data migration script.

Using the above guidelines data migration procedure can be executed effectively and results in the database with concepts that are unique. Data migration script input are the three separate data files, that can be recreated directly from the BELLHOP input or loaded from prepared csv files. These files include SSP vector data, additional bathymetry data (length of segments for variable min/max depths of the water column) and obviously raw BELLHOP output file. These files are first used to create all the entities with their attributes. The designed code presented in the main block of fig. B-1 follows a simple structure: for each entity there is an "inner" function that creates verbal string of a query for a given variable. This string can be used with insert-type query in the entity upload, or with match-type query in relation insert. Completely composed queries for each node, are gathered in a list, which uniqueness is tested with additional functions, before they are send to Grakn in separate transactions.

B-2 Complete Sonar Performance Model Schema & General Query

Listing B.1: Knowledge graph schema written in Graql for the BELLHOP problem.

```
define

##################
## ATTRIBUTES ##
##################

### KGCN attributes

probability_exists sub attribute,
datatype double;

probability_nonexists sub attribute,
datatype double;

probability_preexists sub attribute,
datatype double;

### SCN attributes

scenario_id sub attribute,
datatype long;

num_rays sub attribute,
datatype long;

### Common DEPTH attribute

depth sub attribute,
datatype long; #integer values only

### Bottom attributes

bottom_type sub attribute,
datatype long;

length sub attribute,
datatype double;

slope sub attribute,
datatype long;

### SSP attributes

SSP_value sub attribute,
datatype double,
has depth;

season sub attribute,
datatype string;
```

```
50
51 location sub attribute,
52 datatype string;
53
54 ### Duct\Channel attributes
55
56 grad sub attribute,
57 datatype double;
58
59 number_of_ducts sub attribute,
60 datatype long;
61
62
63 ##################
64 ##  ENTITIES  ##
65 ##################
66
67 sound-propagation-scenario sub entity,
68 key scenario_id,
69 plays defined_by_bathy,
70 plays defined_by_src,
71 plays defined_by_SSP,
72 plays converged_scenario,
73 plays candidate_scenario;
74
75 ray-input sub entity,
76 key num_rays,
77 plays minimum_resolution,
78 plays candidate_resolution;
79 #plays hard_convergence,
80 #plays easy_convergence;
81
82 bottom-segment sub entity,
83 has length,
84 has depth,
85 has slope,
86 plays define_bathy;
87 #plays bathy_src_position;
88
89 source sub entity,
90 has depth,
91 plays define_src;
92 #plays bathy_src_position;
93 #plays supporting_src_pos;
94
95 SSP-vec sub entity,
96 has season,
97 has location,
98 has SSP_value,
99 has depth,
100 plays define_SSP,
101 plays find_channel;
102
```

```
duct sub entity,
has depth,
has grad,
plays channel_exists;

##################
##  RELATIONS  ##
##################

convergence sub relation,
relates converged_scenario,
relates minimum_resolution,
has probability_exists,
has probability_nonexists,
has probability_preexists;

bathymetry sub relation,
relates defined_by_bathy,
relates define_bathy,
has bottom_type;

src-position sub relation,
relates defined_by_src,
relates define_src;
#relates bathy_src_position;

sound-speed sub relation,
relates defined_by_SSP,
relates define_SSP;

SSP-channel sub relation,
has number_of_ducts,
relates find_channel,
relates channel_exists;

candidate-convergence sub relation,
relates candidate_scenario,
relates candidate_resolution;

##################
##    RULES    ##
##################

add-candidate-convergence sub rule,
when {
$scn isa sound-propagation-scenario;
$ray isa ray-input;
not{ (converged_scenario: $scn, minimum_resolution: $ray) isa convergence
    ; };
}, then {
(candidate_scenario: $scn, candidate_resolution: $ray) isa candidate-
    convergence;
};
```

Listing B.2: Query to retrieve example graphs from the Grakn databse.

```
match
$scn isa sound-propagation-scenario, has scenario_id ...;
$ray isa ray-input, has num_rays $nray;
$src isa source, has depth $dsrc;
$seg isa bottom-segment, has depth $dseg, has length $l, has slope $s;
$conv(converged_scenario: $scn, minimum_resolution: $ray) isa convergence;
$srcp(defined_by_src: $scn, define_src: $src) isa src-position;
$bathy(defined_by_bathy: $scn, define_bathy: $seg) isa bathymetry, has bottom_type $bt;
$ssp isa SSP-vec, has location $loc, has season $ses, has SSP_value $sspval, has depth $dsspmax;
$dct isa duct, has depth $ddct, has grad $gd;
$speed(defined_by_SSP: $scn, define_SSP: $ssp) isa sound-speed;
$duct(find_channel: $ssp, channel_exists: $dct) isa SSP-channel;
$sspval has depth $dssp;
{$dssp == $dsrc;} or {$dssp == $dseg;} or {$dssp == $ddct;} or {$dssp == $dsspmax;};
get;
```

Bibliography

[1] https://kpi6.com/blog/interest-detection-from-social-media/knowledge-graph/. Accessed: 27.09.2019.

[2] 3.3. metrics and scoring: quantifying the quality of predictions.

[3] Complete guide to parameter tuning in xgboost with codes in python. https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/. Accessed: 27.07.2020.

[4] Dipolar spreading. https://www.globalsecurity.org/military/systems/ship/images/FIG46.gif. Accessed: 08.10.2019.

[5] Interpretable machine learning. a guide for making black box models explainable. https://christophm.github.io/interpretable-ml-book/ice.html.

[6] Levitus, climatological atlas of the world ocean. https://iridl.ldeo.columbia.edu/SOURCES/.LEVITUS/. Accessed: 08.07.2020.

[7] Networkx: Network analysis in python. https://networkx.github.io/. Accessed: 20.08.2020.

[8] Oc2930: Ocean acoustics. https://www.oc.nps.edu/ bird/oc2930/acoustics/. Accessed: 08.10.2019.

[9] Xgboost parameters. https://xgboost.readthedocs.io/en/latest/parameter.html. Accessed: 08.10.2019.

[10] Ocean Acoustics, Chapter 1. https://www.niot.res.in/, 2013. Accessed: 08.10.2019.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR 2015*, pages 1–15, 2015.

[12] R.T. Bailey. Sound Velocity Corrections for the North Sea Surveyor. *International Hydrographic Review*, 55:124–133, 1978.

[13] A.-L. Barabasi. *Network science*. Cambridge university press, 2016.

[14] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pages 4509–4517, 2016.

[15] Peter W. Battaglia and et al. Hamrick, Jessica B. Relational inductive biases, deep learning, and graph networks. pages 1–40, 2018.

[16] David C. Blair. Information retrieval, 2nd ed. c.j. van rijsbergen. london: Butterworths; 1979: 208 pp. price: $32.50. *Journal of the American Society for Information Science*, 30(6):374–375, 1979.

[17] Michael M. Bronstein, Joan Bruna, Yann Lecun, Arthur Szlam, and Pierre Vandergheynst. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[18] Jaime S. Cardoso. Classification of ordinal data, 2006.

[19] Michael E. Cates and Elsen Tjhung. Theories of Binary Fluid Mixtures: From Phase-Separation Kinetics to Active Emulsions. jun 2018.

[20] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.

[21] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[22] Tianqi Chen. Xgboost tutorials. https://xgboost.readthedocs.io/en/latest/tutorials/model.html. Accessed: 28.02.2020.

[23] Tianqi Chen, Sameer Singh, Ben Taskar, and Carlos Guestrin. Efficient Second-Order Gradient Boosting for Conditional Random Fields, 09–12 May 2015.

[24] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. *CoRR*, abs/1606.07792, 2016.

[25] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 1st edition, 2017.

[26] Noam Chomsky. *Syntactic structures*. Mouton de Gruyter, Berlin New York, 2002.

[27] Wei Chu and S. Sathiya Keerthi. New approaches to support vector ordinal regression. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 145–152, New York, NY, USA, 2005. Association for Computing Machinery.

[28] H. S. Dol, F. Gerdes, P. A. Van Walree, W. Jans, and S. Künzel. Acoustic channel characterization in the baltic sea and in the north sea. *Oceans 2008*, 2008.

[29] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. *CEUR Workshop Proceedings*, 1695, 2016.

[30] V. Červený, M. M. Popov, and I. Pšenčík. Computation of wave fields in inhomogeneous media — gaussian beam approach. *Geophysical Journal of the Royal Astronomical Society*, 70(1):109–128, 1982.

[31] Paul C. Etter. *Underwater Acoustic Modeling and Simulation*. Taylor & Francis, Abingdon, UK, 2003.

[32] M. B. Porter F. B. Jensen, W. A. Kuperman and H. Schmidt. *Computational ocean acoustics*. AIP Press, 1995.

[33] C. Ferri, J. Hernández-Orallo, and R. Modroiu. An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, 30(1):27–38, 2009.

[34] Eibe Frank and Mark Hall. A simple approach to ordinal classification. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2167:145–156, 2001.

[35] Haotian Fu, Hongyao Tang, Jianye Hao, Zihan Lei, Yingfeng Chen, and Changjie Fan. Deep multi-agent reinforcement learning with discrete-continuous hybrid action spaces. 2019.

[36] Stuart Geman, Elie Bienenstock, and René Doursat. Neural Networks and the Bias/-Variance Dilemma, 1992.

[37] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *34th International Conference on Machine Learning, ICML 2017*, 3:2053–2070, 2017.

[38] Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, Jan 2015.

[39] Grakn Team. GRAKN.AI Github account. https://github.com/graknlabs/grakn. Accessed: 14.10.2019.

[40] Grakn Team. KGCN Github page. https://github.com/graknlabs/kglib/tree/master/kglib/kgcn. Accessed: 14.10.2019.

[41] Thomas L. Griffiths, Nick Chater, Charles Kemp, Amy Perfors, and Joshua B. Tenenbaum. Probabilistic models of cognition: exploring representations and inductive biases. *Trends in Cognitive Sciences*, 14(8):357–364, 2010.

[42] Nicola Guarino and Daniel Oberle. Handbook on Ontologies. In *Handbook on Ontologies*. 2009.

[43] Jessica B Hamrick, Kelsey R Allen, Victor Bapst, Tina Zhu, Kevin R. McKee, Joshua B Tenenbaum, and Peter W Battaglia. Relational inductive bias for physical construction in humans and machines. 1, 2018.

[44] Aref Hashemi Fath, Farshid Madanifar, and Masood Abbasi. Implementation of multilayer perceptron (MLP) and radial basis function (RBF) neural networks to predict solution gas-oil ratio of crude oil systems. *Petroleum*, 6(1):80–91, 2020.

[45] Haibo He. Learning from imbalanced data. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 21(9):1263–1285, 209.

[46] Jens M. Hovem. PlaneRay: An acoustic underwater propagation model based on ray tracing and plane wave reflection coefficients. In Michael Taroudakis Papadakis and Panagiotis, editors, *Theoretical and Computational Acoustics*, pages 273–289. University of Crete, Greece, 2007.

[47] Jens M. Hovem. Ray Trace Modeling of Underwater Sound Propagation. In *Modeling and Measurement Methods for Acoustic Waves and for Acoustic Microdevices discussed*, pages 573–600. 2012.

[48] Jens M Hovem and Hefeng Dong. Understanding ocean acoustics by eigenray analysis. *Journal of Marine Science and Engineering*, 7(4), 2019.

[49] Jens C. Huhn and Eyke Hullermeier. Is an ordinal class structure useful in classifier learning? *International Journal of Data Mining, Modelling and Management*, 1(1):45–67, 2008.

[50] J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *IMS 1999 Reitz Lecture*, 1999.

[51] Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition. *IEEE SIGNAL PROCESSING MAGAZINE*, 2(November), 2012.

[52] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[53] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. pages 1–14, 2016.

[54] Wilhelm Kirch, editor. *Level of MeasurementLevel of measurement*. Springer Netherlands, Dordrecht, 2008.

[55] Alex Krizhevsky and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25:1–9, 2012.

[56] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. (1):1–20, 2015.

[57] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning Deep Generative Models of Graphs. 2018.

[58] Xavier Lurton and Darrell R. Jackson. An Introduction to Underwater Acoustics. *The Journal of the Acoustical Society of America*, 115(2), feb 2004.

[59] Dr. Diana McCammon. Tracing uncertainty through bellhop mathematically. proof of concept.

[60] J L McClelland. The interaction of nature and nurture in development: A parallel distributed processing perspective. *International Perspectives on Psychological Science, Volume I: Leading themes*, 1:57–88, 1994.

[61] Deep Mind. Graph net repositoru. https://github.com/deepmind/graph_nets. Accessed: 02.12.2019.

[62] Tom M Mitchell and Tom M Mitchell. The Need for Biases in Learning Generalizations. Technical Report May, 1980.

[63] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[64] Michael B Porter. The BELLHOP manual and user's guide. *HLS Research, , 2010*, pages 1–57, 2011.

[65] Michael B. Porter and Homer P. Bucker. Gaussian beam tracing for computing ocean acoustic fields. *Journal of the Acoustical Society of America*, 82(4):1349–1359, 1987.

[66] David Premack. Human and animal cognition: Continuity and discontinuity. *Proceedings of the National Academy of Sciences of the United States of America*, 104(35):13861–13867, 2007.

[67] David Raposo, Adam Santoro, David Barrett, Razvan Pascanu, Timothy Lillicrap, and Peter Battaglia. Discovering objects and their relations from entangled scene representations, 2017.

[68] Maya Rotmensch, Yoni Halpern, Abdulhakim Tlimat, Steven Horng, and David Sontag. Learning a Health Knowledge Graph from Electronic Medical Records. *Scientific Reports*, 2017.

[69] A. Ruiz-Cañavate and J. Rico. Hourly oceanographic and acoustic variations in the strait of gibraltar, and multibeam echosounder technology. *International Hydrographic Review*, 73(2):108–119, 1996.

[70] Tobias Springenberg Sanchez-Gonzalez Alvaro, Heess Nicolas, Josh Merel, Martin Riedmiller, Raia Hadsell, Peter Battaglia, Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph Networks as Learnable Physics Engines for Inference and Control. *35th International Conference on Machine Learning, ICML 2018*, 10:7097–7117, 2017.

[71] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. Computational capabilities of graph neural networks. *IEEE Transactions on Neural Networks*, 20(1):81–102, 2009.

[72] Samuel Siltanen, Tapio Lokki, and Lauri Savioja. Rays or Waves? Understanding the Strengths and Weaknesses of Computational Room Acoustics Modeling Techniques. *International Symposium on Room Acoustics (ISRA)*, (August):1–6, 2010.

[73] S S Stevens. On the Theory of Scales of Measurement. *Science*, 103(2684):677–680, 1946.

[74] G. Taraldsen, T. A. Reinen, and T. Berg. The underwater gps problem. In *OCEANS 2011 IEEE - Spain*, pages 1–8, June 2011.

[75] Grakn Team. GRAKN.AI Documentation. https://dev.grakn.ai/docs/general/quickstart. Accessed: 14.10.2019.

[76] Chen Tianqi and Carlos Gusterin. XGBoost: A Scalable Tree Boosting System Tianqi. *The Journal of the Association of Physicians of India*, 2016.

[77] W von Humboldt, W F von Humboldt, M Losonsky, P Heath, H W Von, X Yao, K Ameriks, and D M Clarke. *Humboldt: 'On Language': On the Diversity of Human Language Construction and Its Influence on the Mental Development of the Human Species*. Cambridge Texts in the History of Philosophy. Cambridge University Press, 1999.

[78] Nikos Voskarides, Edgar Meij, Manos Tsagkias, Maarten De Rijke, and Wouter Weerkamp. Learning to explain entity relationships in knowledge graphs. In *ACL-IJCNLP 2015 - 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Proceedings of the Conference*, 2015.

[79] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local Neural Networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 7794–7803, 2018.

[80] Peter; Wilcke, Xander; Bloem. The Knowledge Graph as the Default Data Model for Learning on Heterogeneous Knowledge. *Data Science*, 1(November), 2017.

[81] Ruobing Xie, Zhiyuan Liu, Jia Jia, Huanbo Luan, and Maosong Sun. Representation learning of knowledge graphs with entity descriptions. In *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, 2016.

[82] Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *CoRR*, abs/1810.06394, 2018.

[83] Wenhan Xiong, Thien Hoang, and William Yang Wang. DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning. pages 564–573, 2018.

[84] Xilei Zhao, Xiang Yan, and Pascal Van Hentenryck. Modeling heterogeneity in mode-switching behavior under a mobility-on-demand transit system: An interpretable machine learning approach, 2019.

Glossary

List of Acronyms

AI	Artificial Intelligence
AUV	Autonomous Underwater Vehicle
CART	Classification and Regression Trees
CV	Cross-Validation
DSC	Deep-Sound Channel
DL	Deep Learning
GBT	Gaussian Beam Tracing
GLM	Generalised Linear Model
GN	GraphNet
GNN	Graph Neural Network
HP	Hyperparameter
ICE	Individual Conditional Expectation
INDRA	In-house Developed Ray-based Algorithm
KG	Knowledge Graph
KGCN	Knowledge Graph Convolutional Network
ML	Machine Learning
MLP	Multilayer Perceptron
MPNN	Message Passing Neural Network
NLNN	Non-Linear Neural Network
NN	Neural Network
PDP	Partial Dependence Plot
ReLU	Rectified Linear Unit
RMSE	Root Mean Square Error

SLD	Sonic Layer Depth
SMOTE	Synthetic Minority Oversampling Technique
SNR	Signal-to-Noise Ratio
SOFAR	SOund Fixing and Ranging
SPM	Sonar Performance Model
SSP	Sound Speed Profile
TF	TensorFlow
TL	Transmission Loss
TNO	The Netherlands Organisation for Applied Scientific Research
XBT	eXpendable BathyThermograph
XGBoost	Extreme Gradient Boosting