

TBS

Master's Thesis

Nicolas Cuervo-Benavides

Advisor : Dr.-Ing. Holger Jäkel
Supervisor : MSc. Felix Wunsch

Start date : 17th May 2017
End date : 6th December 2017

Declaration

With this statement I declare that I have independently completed the above master's thesis. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis was not previously submitted to another academic institution and has also not yet been published.

Karlsruhe, 06.12.2017

Nicolas Cuervo-Benavides

Abstract

This thesis collects the fundamentals of machine learning and applies them in a determined, state-of-art, communications scenario. CEL thesis rules require it to be about 3-5 pages. It is a summary of what you do in your thesis. Use around 5 pictures and outline whatever you did. And now a few lines of information.

This is a
todo exam-
ple

Polar codes are the first codes to asymptotically achieve channel capacity with low complexity encoders and decoders. They were first introduced by Erdal Arikan in 2009 [Jon05]. Channel coding has always been a challenging task because it draws a lot of resources, especially in software implementations. Software Radio is getting more prominent because it offers several advantages among which are higher flexibility and better maintainability. Future radio systems are aimed at being run on virtualized servers instead of dedicated hardware in base stations [Jon05]. Polar codes may be a promising candidate for future radio systems if they can be implemented efficiently in software.

In this thesis the theory behind polar codes and a polar code implementation in GNU Radio is presented. This implementation is then evaluated regarding parameterization options and their impact on error correction performance. The evaluation includes a comparison to state-of-the-art Low-Density Parity-Check (LDPC) codes.

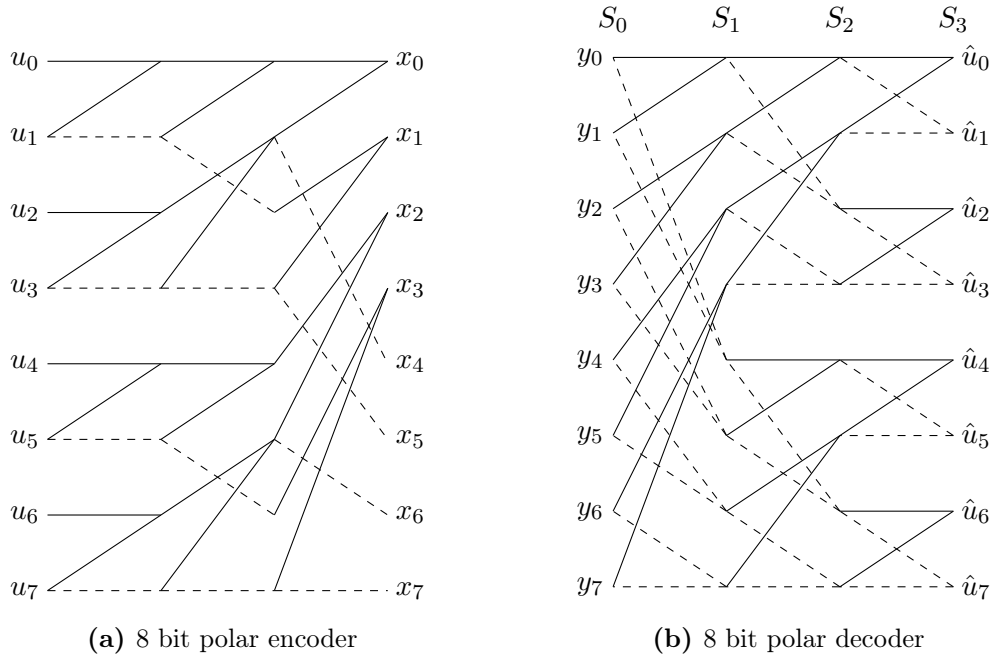


Figure 0.1.: Polar code encoding and decoding

The polar encoder is shown in Fig. 0.1a.

Contents

1. Introduction	3
2. Artificial Intelligence	9
2.1. Overview	9
2.2. Machine Learning	9
2.2.1. The learning problem	11
2.2.2. Types of learning	13
2.2.2.1. Supervised Learning	13
2.2.2.2. Unsupervised Learning	14
2.2.2.3. Reinforcement Learning	14
2.2.3. Training & testing Models	15
2.2.4. Influence of Dataset size in the learning process	15
2.2.5. Data preprocessing	16
2.2.6. Model Evaluation	16
2.2.7. Machine Learning algorithms	17
2.2.7.1. K-nearest Neighbors	17
2.2.7.2. Support Vector Machines	17
2.2.7.3. Binary trees	17
2.3. Deep Learning	17
2.3.1. Neural Networks	17
2.3.2. Convolutional Neural Networks	17
3. Testbed Implementation	19
3.1. Development Tools	19
3.1.1. GNURadio	19
3.1.2. Universal Software Radio Peripheral	21
3.2. Machine Learning models in Python and Jupyter	23
3.2.1. Scikit-learn	23
3.2.2. Keras	24
3.3. Setting up development environment	24
3.4. Data set Generation	26
3.4.1. Measure Campaign	26
3.4.2. Feature Engineering	28
3.4.3. Spectrograms generation	31
4. Evaluation and Results	33
4.1. Performance metrics	33
4.2. Scenario Classification	35
4.3. Dyspan setup comparison	40
5. Live implementation in GNURadio	43

6. Conclusion	45
A. Abbreviations	47

1. Introduction

Back in 1999, Joseph Mitola III coined the term Cognitive Radio (CR)[MM99] as a way to enhance the Software-Defined Radio (SDR) capabilities by the means of a dynamic model that, based on human intervention, improved the flexibility of devices by making them fully configurable and capable of adapting to the communication system's needs, suitable to react to the changes in the surrounding environment. A formal definition for the CR concept provided at [Hay05] encloses the term nicely by describing it as a wireless system that is *intelligent and aware of its surroundings*, whilst being able to learn, adapt and react to changes in the environment, by modifying its operation parameters such as the transmission power, the modulation scheme and its carrier frequency in real-time. Analogously, Jondral [Jon05] adopts the short definition for CR as "an SDR that additionally senses its environment, tracks changes, and possibly reacts upon its findings", becoming an autonomous unit with the potential of using the spectrum efficiently.

CR systems are intended to be immerse in a network, where it interacts with other systems that could be cognitive or non-cognitive radios. According to [GAMS], CR is grouped under three paradigms: underlay, overlay and interweave. The *Underlay Paradigm* allows the CR system to operate under acceptable levels of interference, determined by an interference threshold. Here, the CR is commonly called a Secondary User (SU), providing priority to the other systems in the network which it should not significantly interfere, known also as Primary User (PU). In the *Overlay Paradigm*, the cognitive transmitter knows information about the other transmitters in the network, such as their codebooks and modulation schemes. In addition, this model assumes that message that is being transmitted is known by the CR when transmission by a non-cognitive system is initiated. This provides the cognitive system with multiple choices on how to use this information: for instance, it can be used to mitigate or completely cancel a possible interference happening in the network during transmission. Additionally, the cognitive system could also retransmit this message to other non-cognitive systems in the network, acting as a relay and, effectively, assist increasing the Signal-to-Noise-Ratio (SNR) of the non-cognitive system to a level equivalent to the possible decrease due to CR transmissions. The *Interweave Paradigm*, or opportunistic communication, identifies temporary space-time-frequency gaps where it can intelligently allocate its transmission, increasing the available resource utilization and minimizing the interference with other active users. Hybrid schemes are also actively being developed [WN07] [KWS⁺15] [WKM⁺17], where characteristics from different paradigms are combined in order to achieve an effective use of the available communication resources.

The main characteristic required to apply any of the aforementioned paradigms is awareness, being it in regard of location, spectrum, time, etc. Awareness is achieved by the means of *the cognition cycle* [MM99], which can be seen in Fig. 1.1, which enfolds the way the CR parses the stimuli from the outside world in order to plan accordingly the proper reactions. This cognition cycle revolves around the following concepts

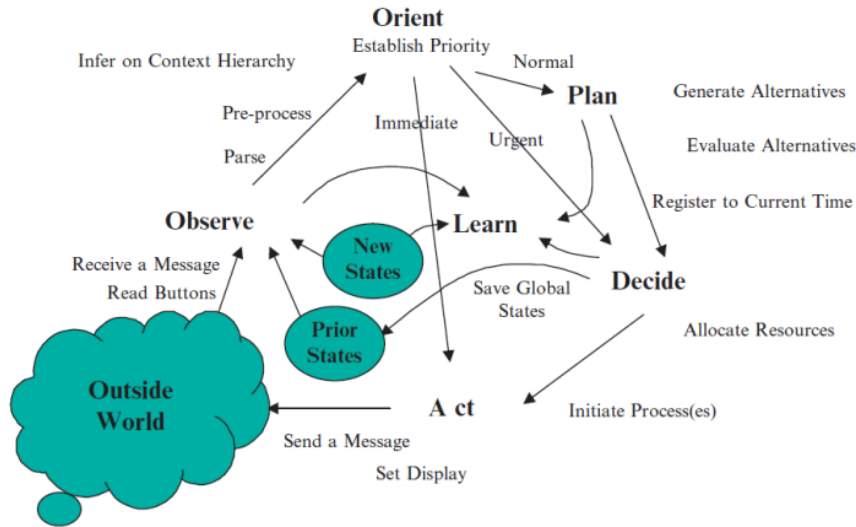


Figure 1.1.: The cognition Cycle[MM99]

- **Observation:** the CR receives any signals from the external world, which can contain any type of information that the system can use in its favor and the favor of a better use of its resources.
- **Orientation:** The CR determines the priority from the received signal as well as the type of reaction based on it.
- **Planning:** results from a normal-level priority, where a plan is generated and the sequence of actions to be taken are established.
- **Decision:** selects among the plan candidates the best proposal and allocates the necessary resources for its carrying-out.
- **Acting:** initiates the decided processes.
- **learning:** is an integration of observations and decisions, based on past and current states that are compared with expectations. When an expectation is met, the system achieves effectiveness. When not, observations are recorded and kept for further learning.

These aspects of CR come in handy when trying to solve one of the current major issues of communication systems: Spectrum Scarcity. The access to radio spectrum is highly regulated by government agencies such as the U.K. Offices of Communications (OfCOM), the Federal Communications Commission (USA) (FCC) and the International Telecommunications Union (ITU), and its access has been historically granted to the highest bidder on so-called *Spectrum Auctions* [Jon05] [SW14]. Therefore, the seek of new technologies that allow a more efficient access to the spectrum is paramount. In an effort to find effective solutions for this increasing issue, the Institute of Electrical and Electronics Engineers (IEEE) created a Standards Committee back in 2005 which, in association with the IEEE Communications Society (ComSoc) and the IEEE Electromagnetic Compatibility Society (EMC) dealt with the generation of standards for dynamic spectrum management. This committee

was dissolved between 2007 and 2010 and, after organizational restructuring, the functions of standardization and spectrum management was handed to the Standards Coordinating Committee 41 (SCC41) - Dynamic Spectrum Access Networks (DySpan) [IEE15]. As part of this efforts to motivate state-of-art research in this regards, DySpan has organized since 2007 the *IEEE International Symposium on Dynamic Spectrum Access Networks* [Com]. Additionally, DySpan has embolden the healthy competition since 2015 by introducing the *Spectrum Challenge*, consisting on inviting team worldwide to solve a problem related with dynamic access to the spectrum and 5G implementations. The participating teams are given a set of requirements and limitations, but are encouraged to push this limits with creativity and innovation. The Karlsruhe Institute of Technology (KIT), represented by the Communications Engineering Lab (CEL), has taken part in these competitions achieving outstanding results, being awarded with the *Subjective Winner* award on 2015 [KWS⁺15] and the *Best Overall Solution* on 2017 [WKM⁺17]. This thesis utilizes the setup used at the 2017 spectrum challenge as base testbed. Fig. 1.2 shows the main characteristics of this setup.

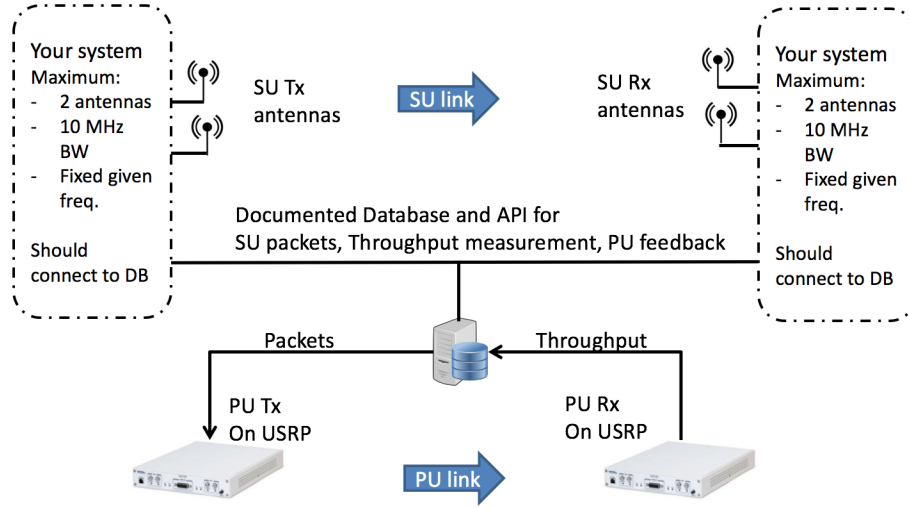


Figure 1.2.: The DySpan Spectrum Challenge Setup [Cha]

By using this configuration and keeping the hardware and overall physical considerations (such as Bandwidth (BW), number of antennas and central frequency), the idea of the challenge was to achieve the maximum throughput between the proposed SU systems, while interfering as little as possible with the existing PU. The competition consisted of two phases: during phase one the situational awareness of the proposed CR system is put under test, as it need to correctly identify the set of PU transmission parameters:

- Bandwidth and Carrier Frequency: along the 10MHz of maximum BW divided in four subchannels of 2.5MHz each, it needs to be detected if the PU is using one, two or four channels for its transmission. Effectively, it is needed to determine which frequencies are being used (identify the frequency hopping pattern) and when are they being used.
- Packed length: the PU transmitter sends packets in a bursty fashion to the corresponding receiver using packets of 100 or 1000 bytes.

- Inter arrival time between packets: the time between a packet transmission might vary from a situation to another. This times could be deterministic for some scenarios, as well as stochastic for others following a Poisson distribution. Correctly identifying the length of the packet, as well as the inter packet time of the current situation, allows to effective opportunistic access to the spectrum.

With this characteristics, a set of 10 different scenarios is built, whose parameters are depicted in Table 2.1

Scenario	Description
0	Single random channel, deterministic interpacket delay of 5ms
1	Single random channel, deterministic interpacket delay of 10ms
2	Two random channel hopping, deterministic interpacket delay of 5 ms
3	Four random channel hopping, deterministic interpacket delay of 10 ms
4	Two random channel hopping, deterministic interpacket delay of 5ms
5	Four synchronous channels, deterministic interpacket delay of 5ms
6	Four synchronous channels back-to-back, deterministic interpacket delay of 2ms
7	Four asynchronous random channels, Poisson distributed interpacket delay with mean of 20ms
8	Four asynchronous random channels, Poisson distributed interpacket delay with mean of 10ms
9	Four asynchronous random channels, Poisson distributed interpacket delay with mean of 5ms

Table 1.1.: Scenario description

The second phase of the competition regards the benchmark of the performance of the proposed SU implementation, where aspects such as innovation of the used waveform, machine learning algorithms used, and opportunistic access to the spectrum were considered. The proposed solutions, including the one proposed by CEL, can be found at [WKM⁺17], [PST⁺17], [PSK⁺17] and [LMH⁺17], where a high level of innovation and state-of-the-art research is compiled.

Being clear that *awareness* has been a primordial characteristic of CR since the conception of the concept until now, understanding that this is an area that invites to further research and considering the uprising research in the field of Artificial Intelligence (AI) algorithms, this work focuses on the learning aspect of CR, using the setup from Fig. 1.2 in order to effectively identify the scenarios described at Table 2.1. Previous research on this field covers aspects such as modulation recognition [OCC16a][OCC16b], resource allocation [ZMJ16], autoencoding and optimization of MIMO systems [OEC17], dynamic spectrum management [Hay05] and context awareness [PSK⁺17][WPR⁺17].

The outline of this thesis is as follows: an introduction to AI, focused on Machine Learning (ML) and Deep Learning (DL), alongside the most used algorithms used in academic and industrial fields is presented in chapter 2. General techniques to avoid phenomena such as underfitting and overfitting of the ML are, as well, portrayed. Chapter 3 describes the details of the testbed set up, the measurement of the data and the implementation of the machine learning models. The evaluation of the learning models is then presented in 4

with metrics of performance. The models are put into a live implementation, where the performance of the algorithms is put into test by classifying the scenarios of Table 2.1 in real-time - This is presented in chapter 5. Lastly, the conclusions and future work are summarized in chapter 6.

2. Artificial Intelligence

2.1. Overview

Intelligence as a concept has been a topic of exhausting research in fields such as neurology, philosophy, neuroscience, neurobiology, datascience, among others. The Oxford dictionary defines intelligence as "*the ability to acquire and apply knowledge and skills*" [Oxf_a]. The first part of this definition applies to what is known as "learning", which is according to the accepted definition of the term as well [Oxf_b], and that supports, from the etymology, the importance of the process of learning on intelligence.

Jeff Hawkins, a dedicated neuroscientist and author, has approached the subject from the engineering and medical flanks, analysing the structure of the brain and having the perspective of the possibilities of replicating artificially the most sophisticated type of intelligence found on Earth: the human. In his book *On Intelligence* [HB04], he captures his findings after inspecting the brain cortex and making a parallel between humans and machines. According to Jeff, "*it is the ability to make predictions about the future that is the crux of intelligence*", and these predictions are based on the experiences from which the intelligent being has learnt, making decisions that lead it to the best possible known result. In order to create artificially a so-called *intelligent agent*, scientist have put extensive effort first on trying to replicate the known intelligence [Bro91][RPP07][Haw], taking the approach of generating a machine that is human-like and that behaves like one, being able to observe its surroundings, learn from stimulus that come from the real world, adapt to changes in those surroundings, plan accordingly to foreseeable process (therefore, make predictions), make decisions and act appropriately. These are the characteristics that Mitola [MM99] described in the cognition cycle for Cognitive Radio (CR), which can be applied to any intelligent agent and, consequently, motivate the further research of Artificial Intelligence (AI).

AI, however, encircles a variety of disciplines that are in themselves a complete course of research, as it can be seen in Fig 2.1. This work focuses only in the top branch: machine learning. However, given the slight differences regarding implementation, a separate section will be dedicated solely to deep learning.

2.2. Machine Learning

Machine Learning (ML) encloses the process of taking a data set that represents any phenomena and learning from it. Any type of being that is capable of learning from previous experiences is showing a kind of intelligence, as it interiorizes the stimulus/data and reacts accordingly when it presents itself again. The vast majority of living beings have this capacity, being the humans who have the lead on its effectiveness. Identifying objects, speaking languages, and reacting to any sensorial stimulus is a result of a successful learning process.

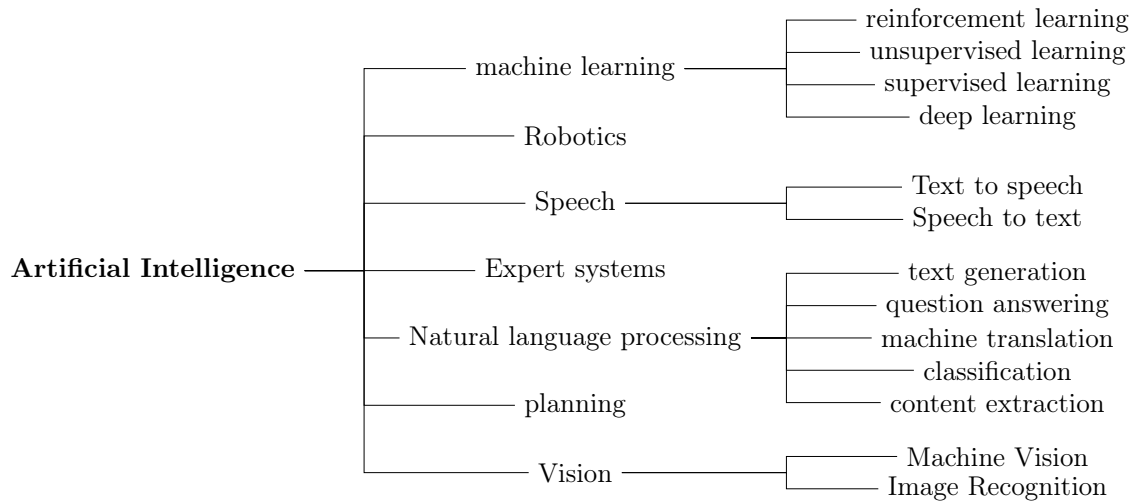


Figure 2.1.: Artificial Intelligence

Generally speaking, learning from data is done when there is no analytic solution to an encountered situation, but there is enough data to adapt to it, generating an empirical solution to a problem that cannot be mathematically a-priori described, but that follows a specific pattern[AMMIL10]. Just as humans do, the idea of machine learning is to generate intelligent agents computationally - teach computers to learn. The idea is as follows: a machine learning algorithm is given a set of data from which it can extract specific information that tells it the specifics about the data. With enough information, the computer is able to make predictions about other data in a different point of time if this data presents the same characteristics.

Although there is no specific mathematic representation of the specific problem to solve, many ML algorithms rely heavily on mathematic definitions and optimization theory. Further information regarding ML algorithms can be found in section 2.2.7. Yet is this versatility provided by the fact of not needing to pin down the specific analytic description of the problem which has impeded this methodology into several fields of knowledge, being nowadays applied to solve problems such as financial forecasting[BM01], medical diagnosis[Kon01], entertainment[BL07] and communications systems (such as this thesis), among others. Examples of everyday problems that are suitable for ML implementation are:

- Ranking links and clicks for a better web search engine and advertisements.
- Custom user recommendations based on purchases/rents/views.
- Prediction of markets and stock exchange.
- Dating sites with reevaluation of algorithms based on successful matches.
- Financial fraud detection.
- Supply chain optimization

- Biotechnology research acceleration by sequencing and screening of DNA and protein/compound structures.
- National security based on enormous surveillance data.

2.2.1. The learning problem

Learning from data is definitely a hot topic, which can be seen from the increasing amount of research and application that has been handed over this theory and methodology. Additionally, it is noticeable how the term has been capturing the mainstream interest and is somewhat heard-of, as it can be seen in the Fig. 2.2, where this trend over the past few years is clear. At this point, it is preeminent to clarify what is the purpose of ML, and when it plays an important role. Although ML has shown to perform outstandingly into solving many problems, it is not intended to move aside the many and well designed analytic solutions for many of the scientific existing problems, but to come in handy when that analytic solution does not describe completely the problem or does not exist. In his book [AMMIL10], Prof. Yaser nicely states that although many problems can be solved effectively using a learning approach or an analytic approach, the point of learning is not to compare itself and overcome the performance over the mathematical description of existing problems, but to be a complementary tool for scientist in their eagerness to solve complex problems without being stuck when facing the lack of a complete description of it.

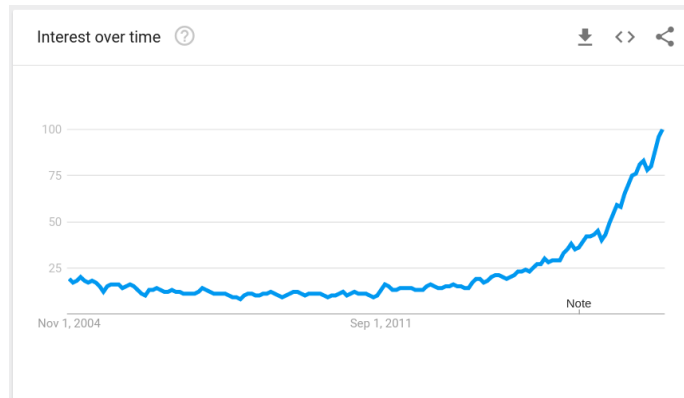


Figure 2.2.: 'Machine learning' Google search trend [Goo17]

The learning problem is summarized in the Fig 2.3. The learning algorithm \mathcal{A} receives data of any form, and its solely purpose is to identify mechanisms that describe that dataset \mathcal{D} closely. This dataset is defined as input-output samples for the supervised learning, input-weights samples for reinforcement learning and only inputs for the unsupervised learning. Further information regarding supervised, reinforcement and unsupervised learning can be found in section 2.2.2. For the sake of the explanation, lets take the supervised learning case, where the dataset \mathcal{D} includes samples of the form $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)$, where x is the input that belongs to the input space \mathcal{X} , y is the corresponding output such that $y_n = f(x_n)$, and belongs to the output space \mathcal{Y} . Now, the learning algorithm \mathcal{A} needs to find that function $f(x)$. For this, it counts with an hypothesis set \mathcal{H} , which are the mathematical representations that the algorithm uses as tools to accomplish his purpose. From \mathcal{H} the algorithm takes one hypothesis $g : \mathcal{X} \rightarrow \mathcal{Y}$ that approximates f . After a g has been

selected, the process estimates how alike the outputs from $g(x)$ are to $f(x)$, and feedbacks an error measure $E(g, f)$. This process is repeated iteratively until an hypothesis produces an acceptable minimum error.

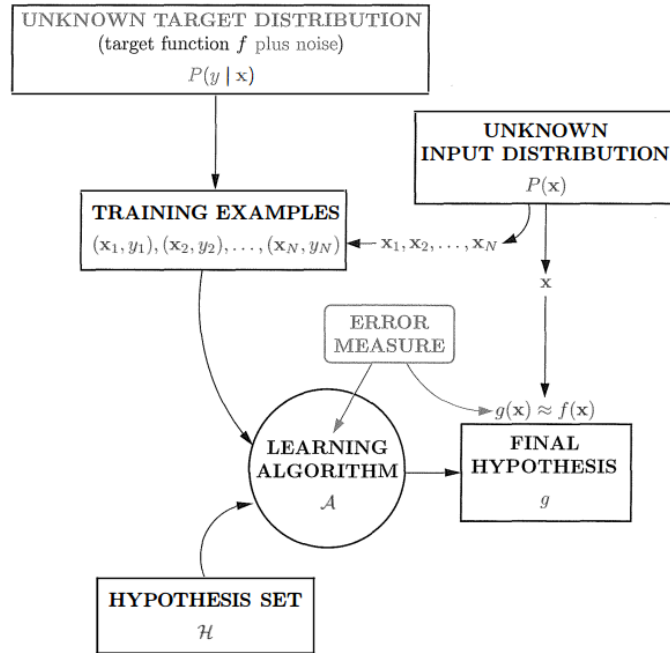


Figure 2.3.: The learning problem [AMMIL10]

Now, it is imperative to determine quantitatively what exactly *acceptable* entails. Different applications can have different tolerances to error, and this affects directly the hypothesis \hat{g} that is chosen at the end of the learning process. This means that this is an end-user parameter that has to be set as a requirement for the whole system.

After taking many different samples from \mathcal{X} , and reducing the error, we should take into account the samples that *we did not take*, meaning the samples that are not in our input set, and that probably behave similar to the \mathcal{X} set - i.e. we should be able to identify similar samples in order to be able to predict. Therefore, a probability distribution is added to both the input samples and the final hypothesis in order to infer from \mathcal{D} the behaviour of samples that are not in \mathcal{D} . Let us assume a binary classification problem, where \mathcal{D} contains two classes A and B in a possibly infinite number of them. From a random sample pick the probability that the input sample is of the A type will be denoted as μ and, consequently, the probability that the input sample is of the B type is $1 - \mu$. The value of μ is unknown and will continue to be unknown in the process. From a random pick of N samples from \mathcal{D} , there is a proportion of ν samples of the A type, and we intend to determine how ν relates to μ . In statistical jargon, we want to determine how our sample relates to the whole population.

From any point of view, the larger the sample that contributes to ν , the closer the relation it has with μ , but this relation can be quantified using the *Hoeffding Inequality* [Hoe63], which states that for any sample size N ,

$$\mathbb{P}[|\nu - \mu| > \epsilon] \leq 2e^{-2\epsilon^2 N} \quad \text{for any } \epsilon > 0$$

Where $\mathbb{P}[\cdot]$ denotes the probability with respect to the chosen sample, and ϵ can be any positive value chosen by the data scientist, and represents the tolerance of error. This inequality says, simply put, that as the sample size N grows, it is *exponentially unlikely* that the realization ν deviates from μ by more than the tolerance ϵ . It can be clearly seen that the only the size of the sample N affects this bound. Consequently, to achieve a small tolerance ϵ , a large N has to be used.

Additionally, it is important to take into account the intrinsic noise of the systems, which can come from the nature of the input samples, i.e. the samples are not product of a deterministic system, or are immersed into some stochastic variation that can be modeled by noise, and that implies that by having the same input into the system *it is probable* get a different output. This entails a change in the labels of the model. That is, instead of having $y = f(x)$, we take y as a random variable resulting from a probability distribution $P(y|x)$. Accordingly, the input data points are therefore generated by the joint distribution $P(x, y) = P(x)P(y|x)$. With this description of the model, our target function (what \mathcal{A} wants to learn) becomes $P(y|x)$, while $P(x)$ quantifies the importance of the input x in our learning accuracy.

2.2.2. Types of learning

There are three types of learning: supervised-, unsupervised-, and reinforcement learning. Each of them has specific characteristics, which are explained in the following subsections.

2.2.2.1. Supervised Learning

Is the type of learning where, in addition to the input dataset, the explicit correct outputs for those given inputs are given to the ML algorithm for training. There are two types of supervised learning:

- **Classification:** its main goal is to predict a *class label* from a determined set of choices. If the number of choices is two, the model corresponds to a *binary classification*. As it has only two options, it is suitable for problems whose expected answer is of the form "yes/no", "present/not present", "valid/invalid". For a greater number of classes the model corresponds to a *multiclass classification*. Examples of classification are:
 - Determining whether an email is spam or not constitutes a binary classification problem.
 - Identifying the zipcode from handwritten digits on an envelope is a multiclass classification problem.
 - Determine whether a tumor is benign based on size and shape data constitutes a binary classification problem
- **Regression:** its purpose is to predict a continuous behaviour, such as a trend, or a floating-number, and it is this continuity what sets it apart from the classification models. Examples of regression are:

- Predict the value of the stock market
- Determine the expected amount of crops yield from a plantation based on data such as previous yields, weather history, etc.

2.2.2.2. Unsupervised Learning

Unlike supervised learning, here the algorithms are feed with data but not with the expected outputs, which make this type of learning suitable for solving problems to which the output is unknown. The model is then in charge of extracting knowledge from the input data all by itself, without no further instructions. There are mainly two types of unsupervised learning that can be found in the literature [MG16], which are the *unsupervised transformations* and *clustering*.

- **Unsupervised transformations:** are models in charge of creating new representations of the input data, so that it becomes easier to understand and/or to use than the original data. This functionality is used, for example, to reduce the dimensionality of data that consists of several features. In such situation, the model transforms the data into a representation that summarizes the input with fewer features. Another important use of this type of models is finding the overall representation of the input data, such as the topic of a full text, or the sentiment in a short comment.
- **Clustering:** this kind of models group the data into determinate groups that share similar characteristics. This is used, for example, to generate suggestions based on previous purchases/views, or to group pictures from a directory with several images to the ones that contain certain people (and suggest tagging the names on them).

As the models do not know beforehand what type of information they are intended to learn, one of the tasks of the data scientist is to assess that the model is indeed learning something useful. This creates the opportunity for this sort of models to be used for the same data scientist to help them identify certain characteristics of the data that were not obvious for the *human-eye*, and certainly get a different perspective of the data from the ML model point-of-view.

2.2.2.3. Reinforcement Learning

This type of learning has a different approach to the previous two descriptions. Just like unsupervised learning, the model does not receive the expected outputs for the inputs it is given but, in contrast, it receives some output *possible* output along with a weight that states how good of an output it is. The idea behind this is that the model is then penalized when it provides a solution that is not according with the possible output, and is rewarded when it is. Then, the model uses this penalizations and rewards to adjust the type of outputs it generates, and so it eventually learns the correct behaviour for the situations it has been immerse into. This type of learning is similar to the way humans learn, in ways such as being penalized with pain when taking a sip of very hot coffee, or rewarded with winning a game of chess.

In that same manner, reinforcement learning comes in handy in teaching an intelligent agent how to play a game, where it is presented to a plethora of options (which makes it difficult

to be modeled as a supervised learning problem) and has to choose the one that brings it near to victory. The most recent example is AlphaGo [Fu17], an intelligent agent capable of winning the world Champion on a Go game. A similar approach has been followed by IBM's with the Deep Blue chess-playing machine [Hsu99].

2.2.3. Training & testing Models

Training and testing are concepts that, when applied to learning algorithms, are not far away from the definitions that are used in common language when applied to humans. In general, *training* a learning model consists of supplying it with enough data from which it can extract information and create generalizations, and testing consist of providing similar data to the model and identify if the generalizations apply. "Generalization" in this context means inferences, decisions or choices that the model makes on data it has never seen before based solely in the data he has learnt from, and clearly the goal of ML is to have models that are able to generalize as accurate as possible.

However, all learning scenarios depend on data, and this data, being as extensive as it can be, does not contain all the information that the data entitles (should the data contain all possible information, learning from it wouldn't provide any profit and would be pointless). Therefore, the question arises: how can it be determined if the model is indeed learning from data?

The way to tackle this problem is separating the data into two: A part from it to be used for training, and the rest of it to be used for validation/testing. Being the training set and testing set generated from the same whole dataset, it is expected that they share enough characteristics for the model to perform well at generalizing. However, there are cases when the model is too complex, and instead of learning to generalize it *memorizes* the training set, and then performs poorly on the training set and on any unknown data. This phenomena is known as *overfitting*, because the model is fit intimately to the training set as is not able to generalize on new data. That being said, one might think that creating the simplest learning model will never end up in overfitting, which is absolutely true. However, in such cases, the model might not be able to even capture the descriptive characteristics of the data, and ends up being incapable of learning from it. This phenomena is known as *underfitting*. Fig 2.4 relates the complexity of the model with its capability of generalizing from new data.

Therefore, it is clear that the objective model is not too simple or too complex, but somewhere in between; what [MG16] legitimately calls "the sweet spot", where the model has learnt as much as it can from the provided data, and is still able to create accurate inferences from unknown samples, ergo make predictions.

2.2.4. Influence of Dataset size in the learning process

It has been constantly repeated that having data is paramount, but "how much data is required?" is a question that will always be asked, and that, unfortunately, has no concrete answer. The logic answer to it is "use as much data as possible". Data is limited, and generally comes along with the description of the problem itself. What this means is that, in data science, it is common for the learning problem to be stated as "The problem to solve is X, and the data available for it is Y" [AMMIL10]. Asking for more data is, in most cases, impossible, and the data scientist has to work with what it has. However, the way

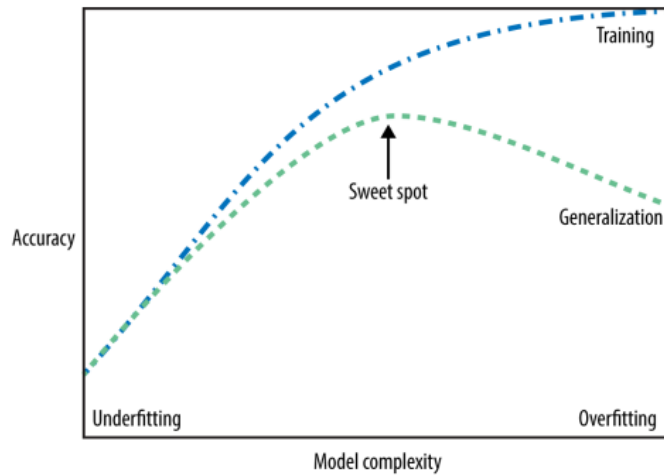


Figure 2.4.: Model complexity vs. Training and test accuracy [MG16]

the scientist extracts information from the data could be the lever that turns a failing model into a successful one.

Moreover, the model complexity is tied to the amount of information that the dataset holds; i.e. if the dataset has considerable variety within, the model can allow itself to grow in complexity without overfitting. This variety usually comes intrinsically with larger datasets, but the clarification has to be made: it is not the size of the dataset what directly yields to variety, meaning that duplicating samples won't help, as repeated data provides no increase on entropy.

2.2.5. Data preprocessing

Possibly the data that is measured or handed to solve a problem does not provide much information *as is*. However, after some careful treatment, that data might be transformed in the specific characteristics that allow an accurate generalization. The process of extracting specific *features* that describe the The main requirement of ML is that data is the main requirement. In the same way, it is necessary to ensure that the data is valid and that information can be extracted from it.

One very interesting aspect about data preprocessing is that unsupervised learning can be used to identify features from a dataset. The most common use cases for this are multiclass visualization, dimensionality reduction and generally finding a representation of the data that is more informative for later processing.

2.2.6. Model Evaluation

There are multiple ways to evaluate the performance of ML algorithms, and they are helpful to figure the model is generalizing from data satisfactorily. In this work we focus on multiclass classification, and one of the most thorough tool for model evaluation is the confusion matrix. When a confusion matrix is calculated for a multiclass classification problem, a square matrix with dimensions equals to the number of classes is generated. Fig ?? shows the confusion matrix configuration for a binary classification, where the classes are "positive" and "negative". Each cell of the matrix contains the number of samples from the

testing set that was classified to that specific class. This means that the diagonal of the matrix states the number of correctly classified elements

Scenario	Description
Positive	Fad
1	Negative

Table 2.1.: Scenario description

2.2.7. Machine Learning algorithms

2.2.7.1. K-nearest Neighbors

2.2.7.2. Support Vector Machines

2.2.7.3. Binary trees

2.3. Deep Learning

2.3.1. Neural Networks

2.3.2. Convolutional Neural Networks

3. Testbed Implementation

This chapter depicts the tools and procedures used in the set up and data preparation prior the machine learning procedures. It includes the steps taken since the start of the work until the moment the first Machine Learning (ML) algorithm started to train. First, an overview of the software and hardware tools is given. Afterwards, a short explanation on the available ML libraries and frameworks is presented, along with the reasoning behind their choosing for this work. Lastly, process of measuring the data and preprocessing it is explained in detail, for the data to be ready to be applied to the learning models.

3.1. Development Tools

As stated in the introduction of this thesis, Software-Defined Radio (SDR) and Cognitive Radio (CR) play an important role in modern communication systems, and is the framework used for most projects at the Communications Engineering Lab (CEL), not only being used as a tool but also being actively contributed to with research results, but also acting as an active agent on open-source improvements. The software and hardware frameworks used are the following:

3.1.1. GNURadio



Figure 3.1.: GNURadio logo

GNURadio [GNU16] is a free and open-source toolkit that provides a large library of signal processing blocks that can be used for several software-defined radio applications. Its functionality does not require a device in the loop, which allows the users to simulate complete communications systems only on a computer. This includes signal sources, modulators and demodulators (such as Phase Shift Keying (PSK) and even Orthogonal frequency division multiplexing (OFDM)), dynamic channel simulators (and virtually any digital filter implementation), math operators, and a plethora of other digital signal processing implementations that have served purposes in academy, research, amateur radio hobbyist and even some government entities. Additionally, thanks to the support for several defined radio hardware [gnu], GNURadio grants the capability of transmitting (given the user has a

rightful license for this purpose) and receive real signals and process them thoroughly.

The usual usage of this software is as follows: the user has a problem or an idea that requires digital signal processing, such as decoding a radio signal or implementing a novel communications' protocol. As said, GNURadio includes several algorithms that serve this purpose, and they are enclosed in so-called blocks. These blocks can then be connected to one another, generating a flow that the signal follows, in a so-called flowgraph, where each block takes a determinate amount of inputs, each input also taking a determinate amount of samples, that undergo the signal processing that the block entitles, and then the block presents its outputs to the next block downstream. If the library provided by GNURadio does not contain implementations that suffice the user needs, new implementations are easily added by the means of a so-called *out of tree (OOT)* module, where the user can provide additional applications, and characteristic that makes the scalability of GNURadio a transparent procedure. Lastly, if the user believes that custom implementation can serve a common purpose and other users, the OOT can be made public following the open source standards, and this way other users can benefit from the same implementation and probably even contribute to it. An extensive collection of OOT that have followed this open source mentality can be found at The Comprehensive GNURadio Archive Network (CGRAN) [CGR].

Regardless of the amount of inputs and outputs on a block, the amount of items required for the algorithm within determines the type of the block:

- If for each output produced the block requires only one input item (1:1), then it is a *sync* block.
- If the block requires N input items in order to generate 1 output item (N:1), the block is a *decimation* block.
- If the block generates M output items for each 1 input item (1:M), the block is a *interpolation* block.
- If the block requires to be extended flexibility, requiring N input items for each M output items produced (N:M), the block is a *general* block.

Most of the blocks are written in a parametrizable fashion, serving multiple purposes with the same implementation by allowing the user to set different settings which can go from the general point of view, such as the vector length of the signal and its data type, to very specific and detailed parameters such as the taps of a filter or the description of a preamble.

The library of algorithms is organized in modules that have a common purpose, and within these modules you find blocks that help achieve that purpose. Examples of such modules are the “gr-qt” module contains the blocks that are intended to be used for visualization purposes using Qt [Qt] and, within this module, blocks such as a “Time sink” and a “Frequency Sink” are found, which are written using Qt and serve as a scope and as a spectrum analyzer, respectively. Another example, more specific, is the gr-channels module, where the user can find different implementations for parametrizable channel simulators, such as fading, frequency selective, and dynamic channel models, among others. Most of these

blocks are written in C++ and python, where each block is, in end effect, a class. In the same programming jargon, the module is a namespace. Therefore, the end user is expected to feel comfortable understanding (and, optimally, using-/writing-) these programming languages in order to be able to use these blocks to the fullest. The interconnection of the blocks, i.e. the flowgraph, is written using Python. For the C++ blocks to be available in the Python interface of the flowgraph, this C++ implementation is translated into Python domain by making use of the *Simplified Wrapper and Interface Generator (SWIG)*. In addition, multiple blocks can be grouped into a single block that serves a specific purpose, and this is called a hierarchical block.

Although coding to the base of the modules and blocks gives the user total control of the details, it is not the only way of getting things done while using GNURadio. The software comes with a Graphical User Interface (GUI) called GNU Radio Companion (GRC), which allows the user to drag-and-drop blocks into a canvas and connect them directly with the ease of a click. Even experienced users grab a hold on this GUI as it provides ease and versatility along with a visible flowgraph that is easy to understand not only for the user but also to other users whose interest has been drawn to a specific application.

In order to use GNURadio, the recommended installation is done by the means of Python Build Overlay Managed Bundled System (PyBOMBS) [PyB], which also allows installation of the modules listed at CGRAN.

3.1.2. Universal Software Radio Peripheral



Figure 3.2.: Ettus Research's logo

One of the SDR devices that is supported by GNURadio is the Universal Software Radio Peripheral (USRP), which is developed and produced by the company *Ettus ResearchTM* [?]. This company has been a one of the most representative suppliers of SDR devices around the globe, and these devices are reknown by its outstanding performance and versatility.

During the complete Dynamic Spectrum Access Networks (DySpan) spectrum challenge competition, three different USRP devices where used for both Primary User (PU) and Secondary User (SU). The PU used for the transmitter and the receiver the USRP X310, which can be seen at Fig. ?? [X30]. This device counts with two wide-Bandwidth (BW) Radio Frequency (RF) daughterboard slots and a large customizable Xilinx Kintex-7 FPGA. Additionally, it has the capability of using high-speed interfaces such as 10gigE and PCIe, with which a maximum of 200MS/s full duplex can be travel through the transport link.

This USRP covers from 10MHz until 6GHz, but based on the daughterboard selected, which serves as RF frontend, the frequency of operation of the device can vary.

As for the SU that was presented by the Karlsruhe Institute of Technology (KIT) CEL group, the transmitter used an USRP N210, depicted in Fig 3.3b [N21]. The N210 has a Xilinx Spartan 3A-DSP 3400 FPGA, and can hold up to 100MS/s through a 1gigE link that connects it to a host machine. This device also requires an RF daughterboard as frontend, for which in the SU implementation the UBX-40, shown in Fig 3.3d [UBX], was used. This daughterboard can operate from 10MHz to 6GHz, providing an instantaneous BW of 40MHz. As for the receiver, a B210, shown in Fig 3.3a [B21], was used. This USRP is a fully integrated, two channel device that operates from 70MHz to 6GHz without the need of additional RF frontend configuration. It provides Full duplex, MIMO (2 Tx - 2Rx) operation up to 56 MHz of instantaneous BW. Furthermore, it counts with a convenient USB 3.0 connection that also serves as power feed.

Although this devices provide high-end performance and its versatility is outstanding, USRP such as the B210 has still a very competitive price for the quality of its elements. Additionally, Ettus ResearchTM is committed with the Open Source community by making its source code available for developers that want to either have a look a it, modify it to add specific functionalities to the USRP devices, or contribute to it.

The interface between the devices and the host machine is performed by the *USRP Hardware DriverTM (UHD)*, an open-source software driver provided by Ettus ResearchTM that furnish support for all USRP devices. This driver supplies API for usage of USRP devices as stand-alone, and also provides support for GNURadio via the gr-uhd OOT.



(a) USRP B210



(b) USRP N210



(c) USRP X300



(d) UBX-40 daughterboard

Figure 3.3.: USRP Devices used in the complete DySpan challenge setup

For this thesis, the SU implementation is reproduced, for which the N210 as transmitter and the B210 as receiver are used.

3.2. Machine Learning models in Python and Jupyter



Figure 3.4.: Python and Jupyter logos

For the learning part of this work, the focus on the implementation was given to a couple of popular Python libraries that have been effective when dealing with ML problems: scikit-learn [SKL] and keras [KER]. These libraries were chosen because of the simplicity of their prototyping as well as their effectiveness when providing an implementation that suits the ML needs by returning fully trained models with exceptional prediction accuracy, and are described in more detail in the following sections. Additionally, the fact that these libraries are written in Python raises interest because this means that they interface optimally with GNURadio, making the inclusion of these libraries transparent, and having the perks of Python such as easy debugging and extensibility.

As for model testing and visualization, Jupyter notebooks [Jup] have been used and are presented as part of the code repository of this thesis. What Jupyter provides is an open-source web application in which a number of interpreters for languages such as Python, R, Julia and Scala are embedded, allowing the creation and sharing of interactive registers that contain code lines and execution output along with visualization fields for plots and graphs and documentation in markdown, in what is called *literate programming*. These notebooks can be shared in multiple formats, such as the native notebook format (for further modification of its contents), as well as HTML, \LaTeX and PDF (generated with \LaTeX). Moreover, it is nicely integrated with GitHub, so that the notebook can be visualized in the webpage of a remote repository without the need of conversion.

Jupyter is the continuation of a long effort for supporting interactive Python interpreters - the IPython Project [IPy]. It has had a fast adoption in the last few years, such that even complete books have been written only using Jupyter as their interface for text editing and code examples. One of the main sources used for this work [MG16] is one example of such.

3.2.1. Scikit-learn

Scikit-learn, formerly scikits.learn and also known as sklearn, is an ML library that features an abundance of algorithms for supervised and unsupervised algorithms, including the ones described in section 2.2.7. This library came as a result from a *Google Summer of Code* project as a third party extension to SciPy, from where it gets its name. This library was used in this thesis for all the learning based on the extracted features listed in section 3.4.2.



Figure 3.5.: Scikit-learn logo



Figure 3.6.: Keras logo

3.2.2. Keras

Keras is a high-level neural networks API written in Python that uses TensorFlow [Ten], CNTK [CNT] or Theano [The] as a backend. The way it is written in a way that allows datascientist to prototype and experiment fast. As per its documentation [KER]: *"being able to go from idea to result with the least possible delay is key to doing good research"*, and Keras certainly intends to keep the coding part as simple as possible for the designer to focus on the idea and not the programming of it. For this, it presents an API that is:

- User-friendly: with ease for writing, reading and understanding.
- Modular: models have a clear begin and end, and they can be easily connected with other models with low to none restrictions.
- Extendible: new functionalities and features are easily added to the mainstream, and the existing codebase is well-documented and exemplified.

For this project, Keras is used for convolutional neural networks implementation, using Tensorflow (with GPU support) as a backend.

3.3. Setting up development environment

Identified the tools that are used in this work, now the installation and set up steps are described. This thesis was completely done using Linux-based operative systems, and the steps depicted below have been tested on Fedora 25 as well as on Ubuntu 16.04LTS. Installation in other Linux distributions have small to no difference from these steps. For the setup on a machine using Microsoft WindowsTM, please refer to the documentation of each of the tools listed in section 3.1.

For GNURadio and UHD, PyBOMBS was used because of the ease it provides on installation, as well as the capability of installing the software in a self contained environment

that does not pollute the system domain. However, installation by source can also be performed, and a very detailed sequence of steps can be found in the following for Linux [Lin], MacOS [OSX] or Windows [Win]. First of all, PyBOMBS needs to be installed, and the recommended way to do so is by using Pip Installs Packages (pip) which, as well, is the recommended way to install Python packages. Installation is simply done by running the following command, which will download the latest release:

```
$ [sudo] pip install PyBOMBS
```

In order to install the latest version from git, the following slight modification to that command is needed:

NOTE: along this work only Python2 was used because of compatibility with the current master branch of GNURadio. Future work includes porting this work onto the Py3k improvements of GNURadio, along with newer versions of the following installed libraries.

```
$ [sudo] pip install [--upgrade] git+https://github.com/  
gnuradio/pybombs.git
```

Now, there is an option for PyBOMBS to parse the best possible configuration specific to the development machine (such as amount of threads used for package installing and a git-cache that speeds up repeatable git clones). This configuration is setup as follows:

```
$ pybombs auto-config
```

PyBOMBS uses so-called recipes that contain the software description as well as its dependencies. This makes possible to automatically generate dependency trees, that will be automatically installed when a package requires them. The recipes are downloaded and installed as follows:

```
$ pybombs recipes add-defaults
```

Up to this point, PyBOMBS has the references necessary to install GNURadio and UHD, and it is only needed to specify where in the development machine this software is to be located. For it, we generate a self-contained prefix, which allows us to keep the software all in one place without polluting the system domain. The next command creates the directory `~/workarea`, gives it the alias "workarea" (which will help PyBOMBS identify this prefix uniquely), and will start the GNURadio installation after installing all its dependencies, UHD included:

```
$ pybombs prefix init ~/workarea -a workarea -R gnuradio-  
default
```

In order to install the ML libraries, there are multiple options as well. The libraries can be installed using pip with the following command:

```
$ [sudo] pip install numpy scipy pandas graphviz jupyter  
scikit-learn tensorflow keras
```

Numpy and Scipy should have been installed during the GNURadio installation, but are listed in the command for completeness' sake. It is also recommended to install this tools in a dedicated Python Environment, such as Conda or Virtualenv. Procedures on how to generate these virtual environments, and already saved environments that can be pulled and installed (included the versions used in this work) are included in the GitHub repository of this thesis [Cue17].

NOTE: For training using Keras/Tensorflow, and given that the development machine has a compatible graphic card, it is recommended to install Tensorflow with GPU support. By doing so, the training times are reduced drastically. This work was developed and written using a Lenovo Thinkpad W541, which counts with a NVIDIA Quadro K2100M Graphic card, compatible with CUDA. For this dedicated installation, a combined procedure from [Yad] and [And] was used.

3.4. Data set Generation

This part of the work regards the steps taken in order to have the data ready for the ML algorithms to learn from it. It covers the testbed setup, the raw data (I/Q samples) measurement, and the data preprocessing.

3.4.1. Measure Campaign

The first step taken was to set up the PU communication link over the air, and recording the raw samples just as the SU would be able to "hear" them. The measurement setup is shown in Fig 3.7, where two parts are labeled separately.



Figure 3.7.: Measurement Setup

The part labeled with ① regards the transmission part, which is going to be sending frames over the air in ten different fashions, described in Table 2.1. The host machine in this part has the connection to the database, from where the information frames are extracted and put as payload of the transmitted frames. Additionally, the GNURadio flowgraph that generates the signal, which can be seen in Fig 3.8, is also hosted and run from this computer. A summary of the path that a frame travels from the database until the transmitter is as follows: in Fig. 3.8 it can be seen that a connection to the database is done in the "Cmd pktgen" block, which is in charge of retrieving the information frames from the database, in form of Protocol data unit (PDU), and feeding them into the signal processing blocks. After converting the PDU into a byte tagged stream (a stream of data that has metadata attached to it in form of tags), the stream flows into the OFDM transmitter block, which is a hierarchical block that allocates the carriers for an OFDM transmission, applies a Fast Fourier Transform (FFT) to them, and appends a cyclic prefix.

Right after, the output, which is a complex I/Q tagged stream, is then converted again into PDU and feed into the packet controller block. This block is in charge of generating

the different scenarios that are set to be identified in this thesis by the means of learning techniques. The parameters of this block are not modified along this work, except one: the "scenarios list". By setting the scenario list to an specific value from 0 to 9, a determinate behavior can be set and recorded, and when this samples (after the feature extraction of section 3.4.2 are paired to the selected scenario, they have the form (input, output), suitable for supervised learning techniques. The rest of the flowgraph, consistent of resampling and multiplying with a sine signal, corresponds to a traditional mixing to the specific channels between the 10MHz BW centered at 3.195GHz. As for the RF frontend, a N210 USRP was used, which in the flowgraph is represented as the "UHD:USRP sink" block.

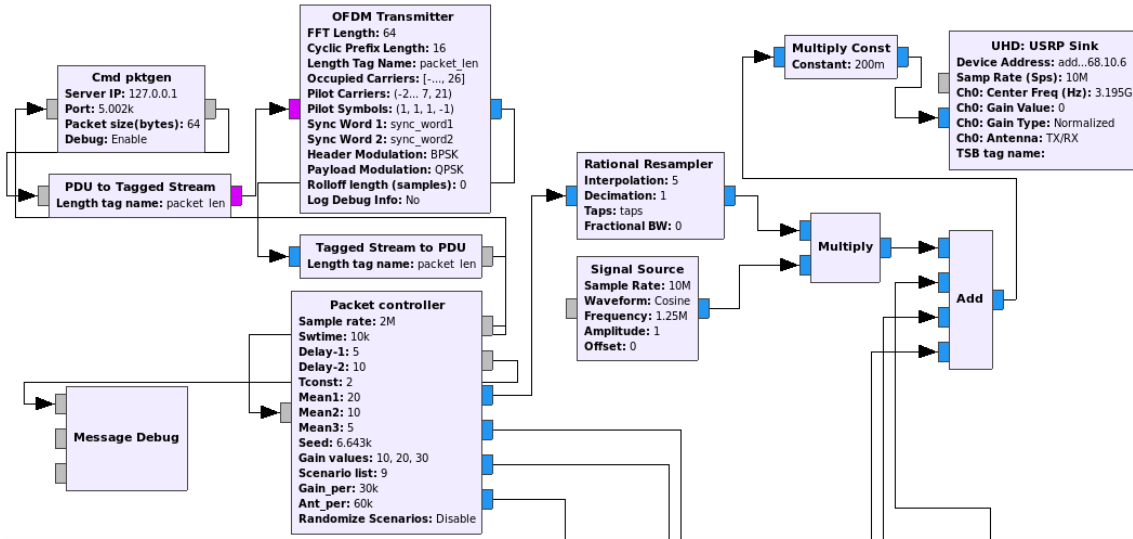


Figure 3.8.: PU GNURadio flowgraph

The part of the Fig 3.7 labeled with ② corresponds to the recording system. In this machine, a very simple flowgraph has been implemented and is shown in Fig 3.9. An "UHD: USRP source" block, which in real life is an USRP B210, is connected directly to a couple of scopes: A Waterfall sink and a Time Sink. These scopes, however, are only used for supervision of the presence of energy over the air, and have no effect whatsoever on the recorded signal. The main path is then the "File sink", which directly saves the data presented at is input into a file, and the "Head block" which serves as gauge to limit the amount of data that is saved to disk. Taking disk space into consideration, it was decided to record 1 minute of raw samples per scenario: 30 seconds with the presence of DC offset in the transmitter, 30 seconds without it. The DC offset is taken into consideration because the assumption is made that there is no strict control on how the PU transmissioin is made appart from the specs given, so this is an attempt to cover every possible situation. The reason behind this time limitation is the amount of data that has to be taken into consideration. The math behind it is as follows:

$$\frac{10 \text{ MS}}{1 \text{ s}} \cdot 60 \text{ s} \cdot \frac{8 \text{ B}}{1 \text{ S}} = 4.8 \text{ GB} \quad (3.1)$$

Each sample is complex valued of 64 bits, having 32-bit for the real part and 32-bit for the imaginary part, totalling 8B per sample. At a rate of 10MSps used at the DySpan Spectrum Challenge, a minute of measurement corresponds to 4.8GB of data. Now, it is of interest to have measurements of different PU signal power, in order to determine

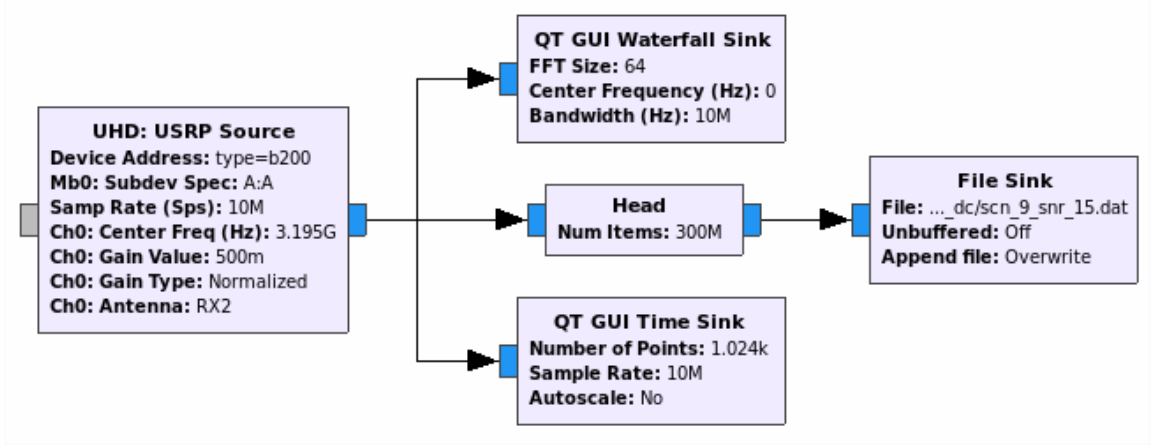


Figure 3.9.: Flowgraph used to record over-the-air samples

the performance of the learning algorithms facing low Signal-to-Noise-Ratio (SNR). It was determined that measurements for SNR values equals to -5 dB, -2.5 dB, 0 dB, 2.5 dB, 5 dB, 10 dB, 15 dB would represent the overall performance of the algorithm facing bad SNR as well as "good-enough" SNR. This means that the measurement was repeated 7 times per scenario.

$$4.8 \text{ GB} \cdot 7 \text{ SNR levels} \cdot 10 \text{ scenarios} = 336 \text{ GB} \quad (3.2)$$

This clearly represents a limitation for the host machine of the lab, that has shared resources with other users and, regardless and in total, has only a total of 500 GB of hard disk. However, this is done this way as an attempt to reproduce the challenge setup closely. Clearly, other methodologies could have been followed, and a short argument regarding this detail is given in chapter 6.

Here is important to state a remark on how the SNR was calculated, and how the control over it was applied to ensure that the samples are independent and that they contain the right information. First, a calibration procedure was made on the PU transmitter, using a handheld spectrum analyzer. Starting with the maximum gain on the transmission chain on the N210, it was double checked that by reducing a value XdB on the gain, a decrease of XdB in the power measurement was also seen. An example for this is shown in Fig 3.10, where a variation of 5dB gain is applied at the transmitter. Additionally, the average received power was also plotted at the receiver side, along with SNR calculations shown in the GUI as labels, as seen in Fig 3.11.

3.4.2. Feature Engineering

Having recorded the raw I/Q samples, it is required to apply signal processing techniques in order to extract features that this data could entitle and that would be representative to identify the scenario that the PU transmission is describing. For this, the implementation that was used in [WPR⁺17] was used. The flowgraph that serves the purpose of feature extraction is shown in the Fig 3.12. In Fig 3.12a, the files recorded from the PU transmission are passed through an FFT, and then the "Energy Detection" block is in charge of determining which channel has an measure of energy greater than the given threshold,



Figure 3.10.: Calibration of gain variation on PU transmitter (5dB variation)

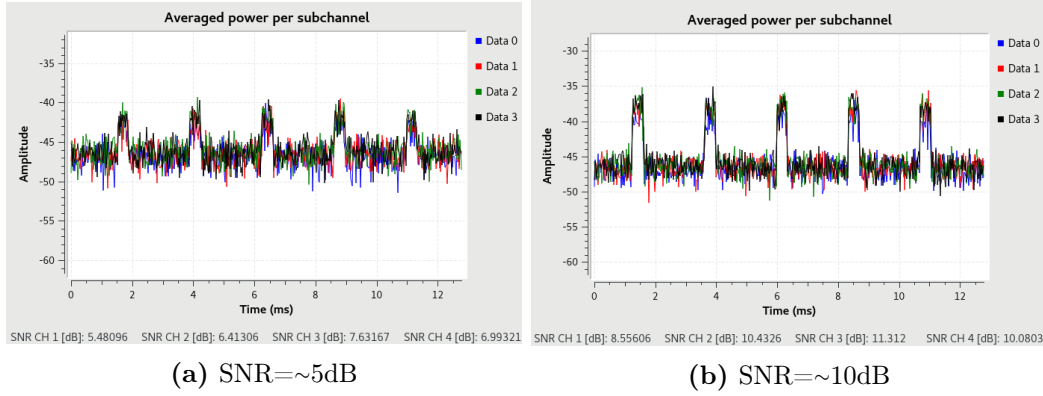


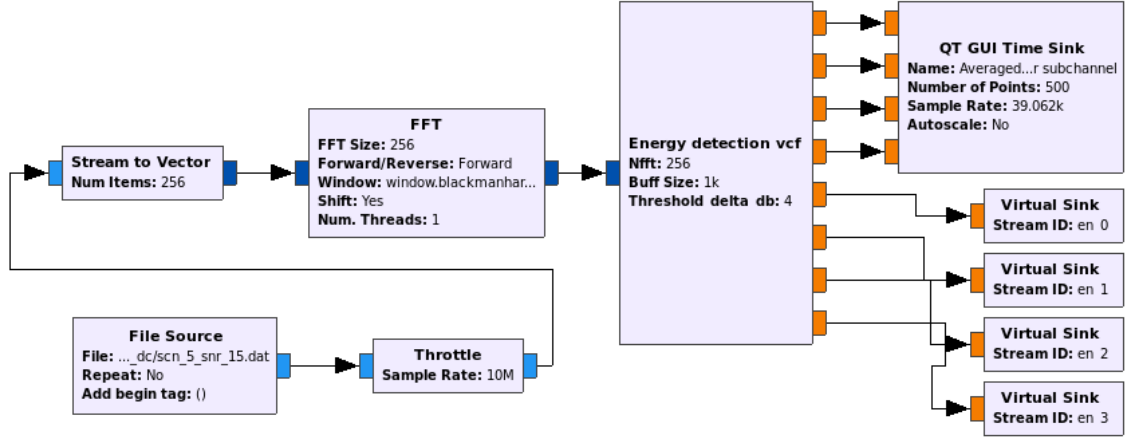
Figure 3.11.: SNR display at receive side

which can be set as a block parameter. This block has 8 outputs: the top four provide the average power per subchannel, useful for plotting. Fig.3.11 was plotted using these outputs; the bottom four send simply ones when the channel is identified as occupied, and zeros if the channel is identified as free.

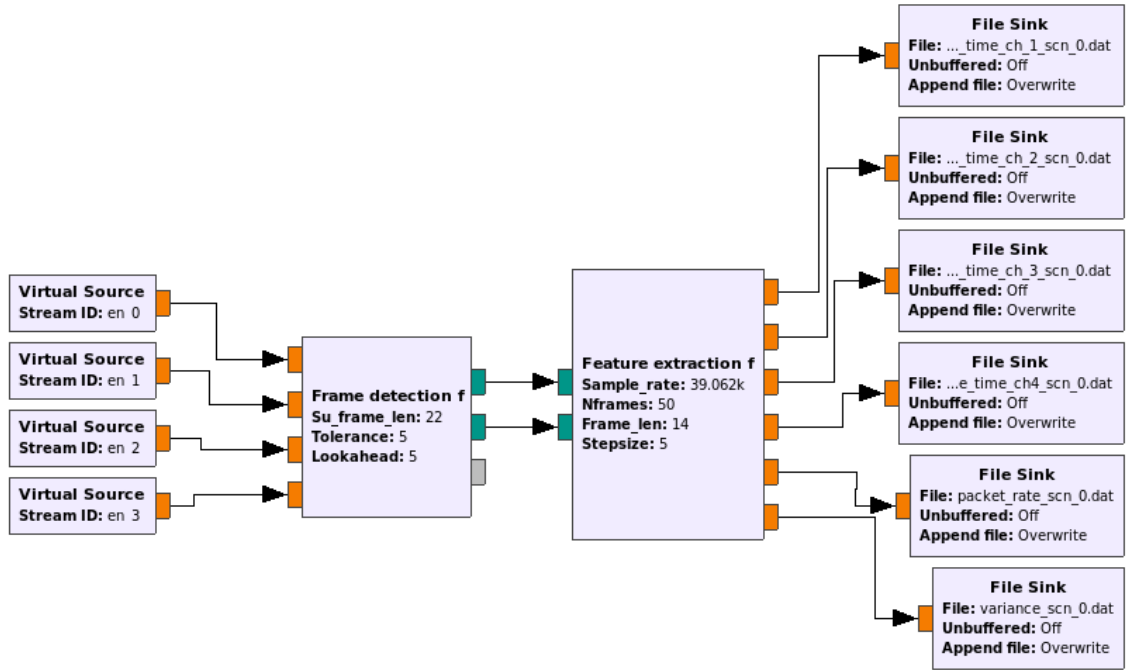
The second part of the flowgraph, shown in Fig 3.12b uses the boolean-like output from the "energy detection" block and analyzes them as *frame events*, identifying the frames over the air and categorizing them as PU frames or SU frames based on its length, and generating these frame events if the analyzed frame per channel corresponds to the PU. Then, these events are taken at the "Feature Extraction" block, which generates three main features based on its inputs:

- the average interframe time, i.e. the time delay between PU frames.
- the packet rate, meaning the amount of packets/frames that are passing through per second.
- the variance of the interframe time, which intends to determine if this interframe time is deterministic or stochastic and, if the later, with which variation.

Although these are effectively only three features, the interframe time is analyzed per channel, which explains the 6 outputs of the "feature extraction" block. This information is then



(a) Energy Detection



(b) Frame Events detection and write features to file

Figure 3.12.: GNURadio Flowgraph for feature extraction

written to disk in different files. This procedure is repeated for all the files saved during the process described in section 3.4.1, generating a total of:

$$10 \text{ scenarios} \cdot 7 \text{ SNR levels} \cdot 6 \text{ feature files} = 420 \text{ files} \quad (3.3)$$

This files are provided in the final Dataset product of this work, and a convenient script that automates the process of the feature extraction is provided at [Cue17]. As stated at the beginning of the chapter, this features will be used for supervised learning, and the results are recorded in chapter 4

3.4.3. Spectrograms generation

The feature extraction process is convenient and works outstandingly as seen in [WPR⁺17]. However, it depends on a good PU power level in order for the frame events to be generated. A way to go around this limitation is applying a different technology, such as image recognition using Convolutional Neural Networks (ConvNet). The idea is to generate images that describe the unique characteristics of each scenario, and feed them to an image classifier. The images that better suit this need are the spectrograms. Based on the work of [PSK⁺17], spectrograms were generated using the GNURadio flowgraph shown in Fig 3.13

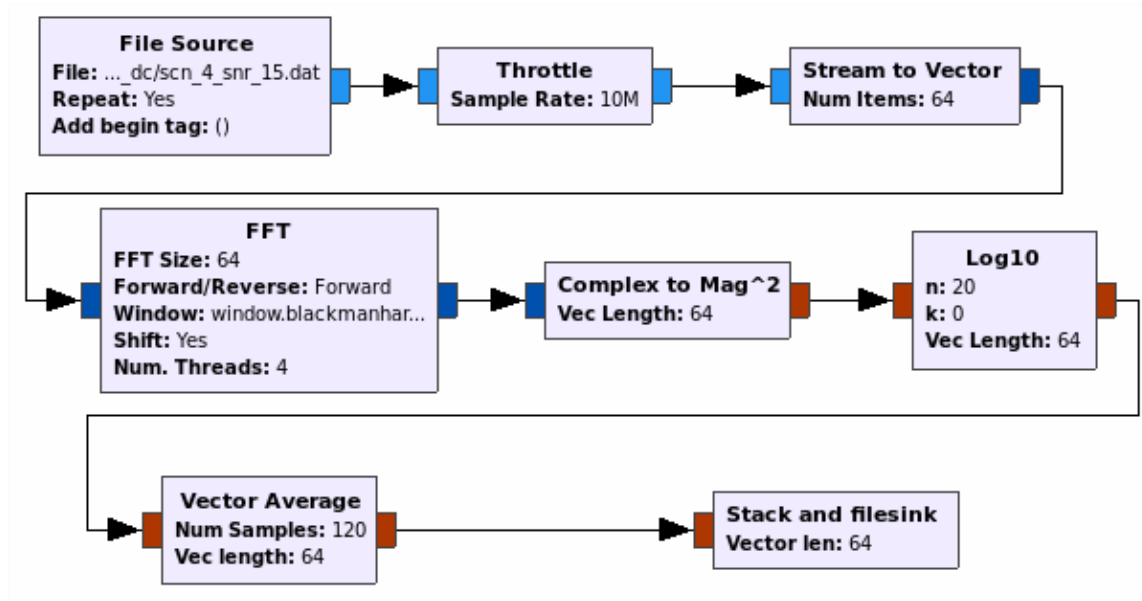


Figure 3.13.: GNURadio Flowgraph for spectrogram generation

Most of the procedure is rather straight forward: the I/Q data files are read and its stream is moved into a vector for the FFT calculation. Our target images are going to have a 64x64 pixels dimension, so that sets as well the FFT size to 64. The following blocks calculate the power of the signal. The last two blocks require a more detailed explanation. The "Vector average" block calculates the arithmetic mean over a certain amount of vectors. For the sake of clarity, it can be assumed that this block operates over a matrix (or a 2-dimensional array) of dimensions NxM, being M the length of the input vectors, set in the block via the parameter "vector length" and set to 64 in this specific case, and N the amount of vectors to operate, set in the block via the parameter "Num Samples" and set to 120, which will become clear shortly.

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1m} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{nm} \end{bmatrix}$$

The block takes 120 vectors and presents at its output the arithmetic mean according to the matrix. After that, the block "Stack and filesink" takes 64 averaged vectors, stacks them vertically and generates an image object that is later saved as a JPEG image to disk.

include a
volk para-
graph in
gnuradio

tikz magic

These images are then used for the image classification. From the files recorded using the procedure described at section 3.4.1, a total of 84700 pictures is generated, from which 90% is used for training the ConvNet and 10% is used as testing set.

4. Evaluation and Results

In this chapter a benchmark of the analyzed algorithms is presented, based on dataset size and complexity. For the first part, the algorithms used for supervised learning are presented, onto which an analysis of the different complexities used is made. From the benchmark, the trained models with the best performance are saved for persistence and then used in the live implementation presented in chapter 5.

4.1. Performance metrics

The impact of the dataset size on the algorithm performance is of special interest. Given that the data recorded using the procedure described in section 3.4.1 has a fixed size, this analysis was performed by slicing the resulting feature extraction files to emulate different dataset sizes. It is important to notice that only slicing for emulate smaller datasets, and not repeating samples for emulation of bigger datasets, has a significant impact in the performance, as noted in section 2.2.4. This is due to the fact that repeated samples present no new information to the model, and there is nothing new to learn from them.

The dataset is, then, sliced into four different sizes:

- Small dataset (S): is one fourth ($1/4$) of the original dataset
- Medium dataset (M): is one third ($1/3$) of the original dataset
- Large dataset (L): is half ($1/2$) of the original dataset
- Extra Large dataset (XL): the whole set of features extracted as recorded from the signals over the air.

Furthermore, a benchmark of the learning algorithms is made based on variations on their complexities. Each algorithm has different ways to represent its own complexity, and this has a significant impact over its overall performance. The Machine Learning (ML) algorithms used in this analysis are: K-nearest neighbors, Support Vector Machines (SVM) and binary tree classifiers. Their complexities are set as follows:

- K-nearest neighbors: the complexity of this algorithm is set by the amount of neighbors that are considered in order to make a classification decision. For the benchmark, different models were trained using 2, 4, 10 and 50 neighbors.
- SVM: this algorithm allows the designer to select the type of kernel that is used during the learning process. The function of the kernel is to generate internally non-linear transformations over the input data while still behaving as a linear classifier. For this work, the Radial Basis Function (RBF) is used as a kernel because of its known good performance on multiclass classification problems at the cost of longer training

times. The complexity of this model is then set by providing different values to the 'C' parameter of this RBF kernel. This parameter sets the trade off between the proneness of the model to result in a erroneous classification and the simplicity of the decision boundary. For this work, values of $C=(1, 1000, 1000000)$ are given.

- Decision Trees: in this model, the complexity is ruled by the depth of the decision branch. Here, depths of 2, 5, 10 and 50 are given.

Moreover, it is also interesting to determine how the models behave when the features have been scaled. A first glance of the impact of a scaler is presented by using the `StandardScaler()` from `SKlearn`, which simply removes the mean of the features and scales them to unit variance. A side-by-side comparison on model performance when trained with unscaled features vs.scaled features is shown as a result from the benchmark. The results are depicted in Figs 4.1, 4.2 and 4.3.

From Figs 4.1, 4.2 and 4.3 a couple of conclusions can be extracted. First, it is seen that the accuracy of the model increases as more data is taken into consideration, which is well according with the theory and serves as queue for applying the same implementation with more data as proposed in chapter 6. Furthermore, three different behaviours are appreciated:

- : As complexity increases, the K-nearest neighbors algorithm shows a decrease on its overall accuracy, which is an indication that the model is overfitting, and suggests that with such a complexity (50 neighbors, in this case), the model is starting to "memorize" characteristics of the training dataset, and is set to perform badly if unknown samples are feed to it, as its capabilities for generalization are in decay. Moreover, the prediction times for the highest level of complexity are already about two times greater than the second highest, making it not suitable for live implementations. Furthermore, this model shows a both a reduction in accuracy as well as an increase in prediction time when the features have been scaled, which simply states that the model performs well without further data preprocessing right after the feature extraction.
- The decision tree model shows a steady increase on its accuracy as complexity increases, without reaching a clear overfitting state. Complexity has a clear effect on the training times, but it is still on the range of the milliseconds - being a process done only once, this does not affect dramatically the model selection and does not closes the possibility of increasing the complexity for this specific use case. This can also be backed on the fact that the prediction times seem to stay constant with this increase. Interestingly, not only the training times are reduced after scaling, but also the prediction times are drastically improved, reaching about 10x faster predictions using the standard scaler. With comparable accuracy, scaling the features seems like a prospect procedure when this model is used in time demanding or live implementations.
- the SVM do not show a clear tendency to improve as complexity is increased, and just like the K-nearest neighbor models, performs quite well in small sizes of datasets. However, the best accuracy for this model is still below the accuracies of the other models regarded in this benchmark. Additionally, the training and prediction times exceed dramatically the times achieved with the other models, being about 8000 longer for

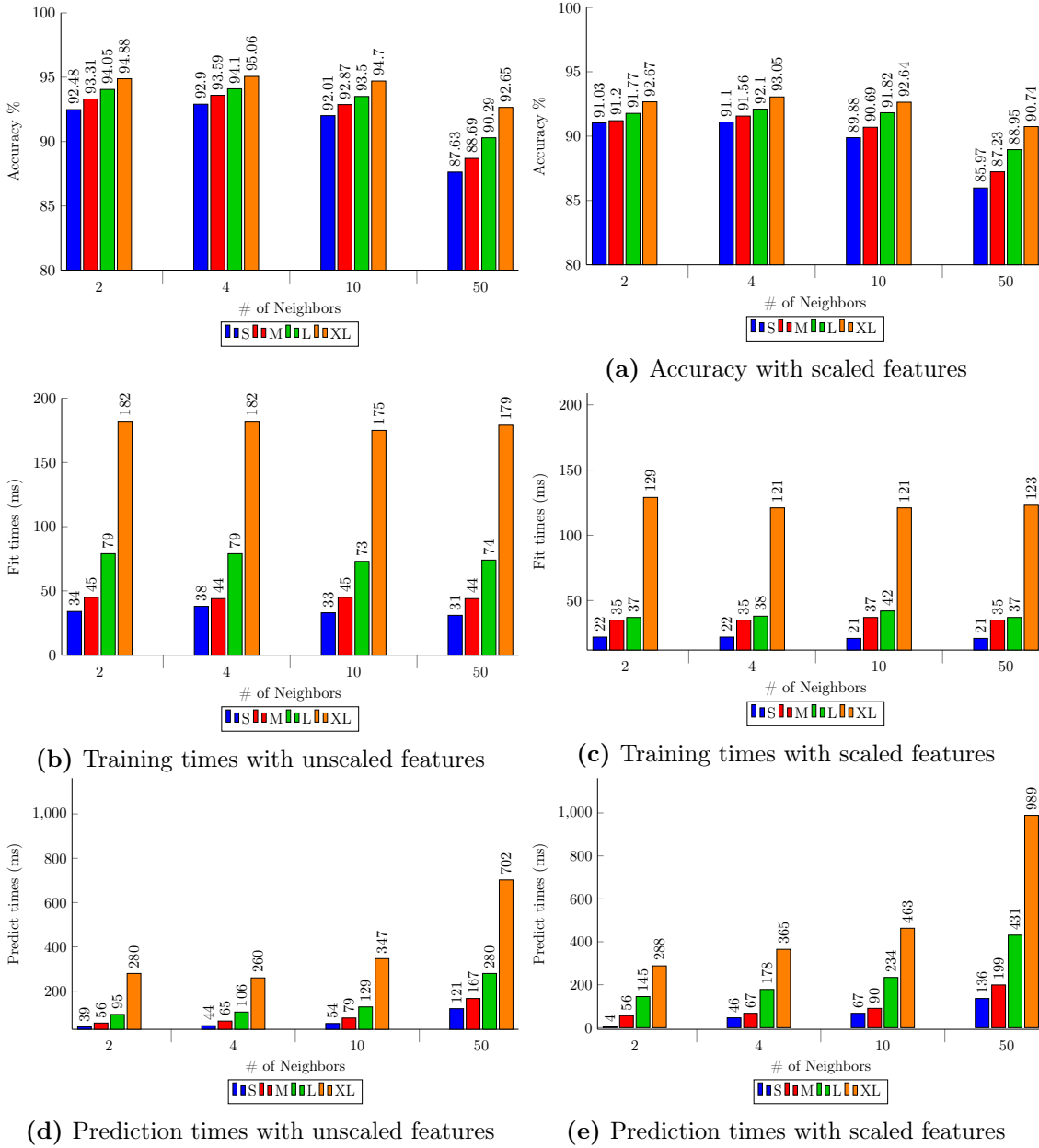


Figure 4.1.: K-nearest Neighbors

training and around 500 times longer for prediction. The fact that this model is that slow will definitely affect considerably the performance of realtime implementations, aspect that is crucial in systems such as cognitive radio.

4.2. Scenario Classification

Besides of the general performance that the learning models have in regard of the whole testing set, it is paramount to determine how good they perform by classifying each of the specific scenarios of Table 2.1. To assess this, confusion matrices are used to determine the

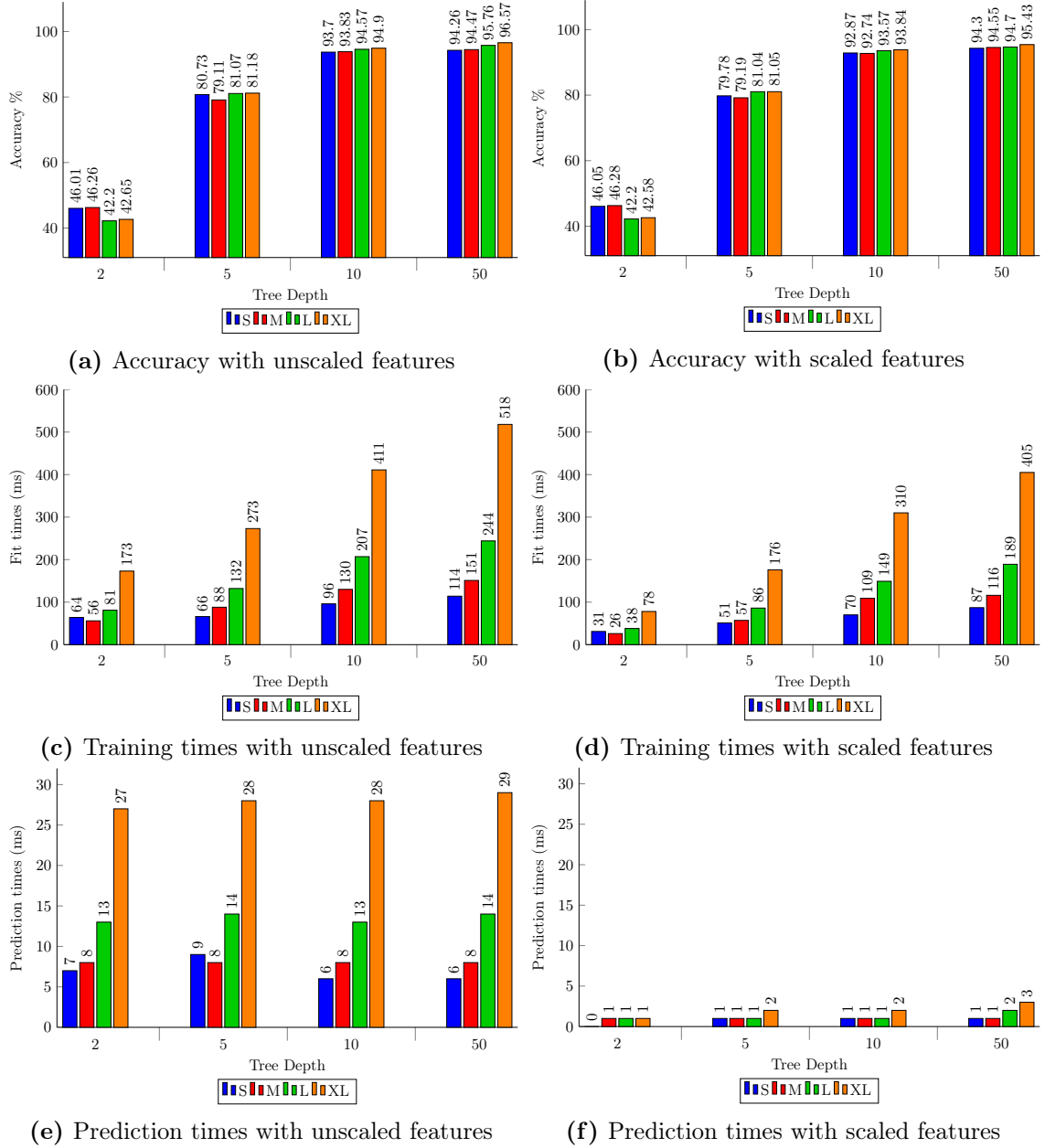
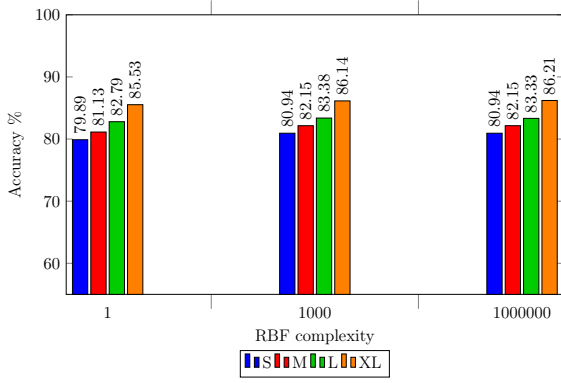
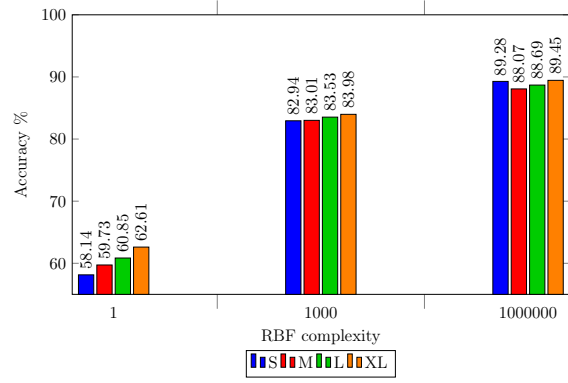


Figure 4.2.: Decision Trees

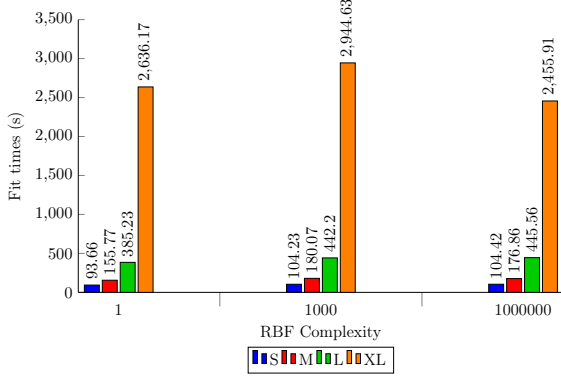
amount of rightfully classified scenarios, along with the false positives and false negatives. On this analysis, it is only of interest the amount of correctly classified scenarios, and any misclassification affects the implementation equally, regardless of its type. Moreover, in section 4.1 can be seen that each algorithm behaves differently in terms not only of accuracy, but also in training time and prediction time. As the models are trained only once, the "training time" does not play a role in the model selection for this work, as it does not have any repercussion on the performance of the model when new values are applied to it. Therefore, "prediction time" and "accuracy" are metrics of quality that are considered for this models on its selection to be applied on real time scenarios.



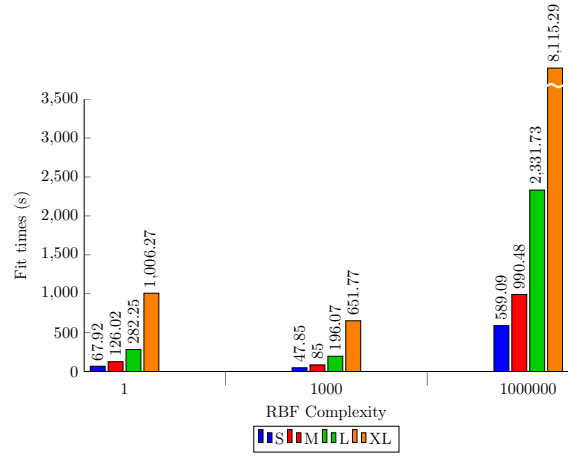
(a) Accuracy with unscaled features



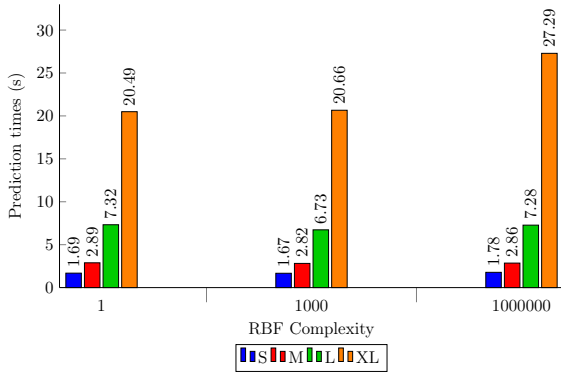
(b) Accuracy with scaled features



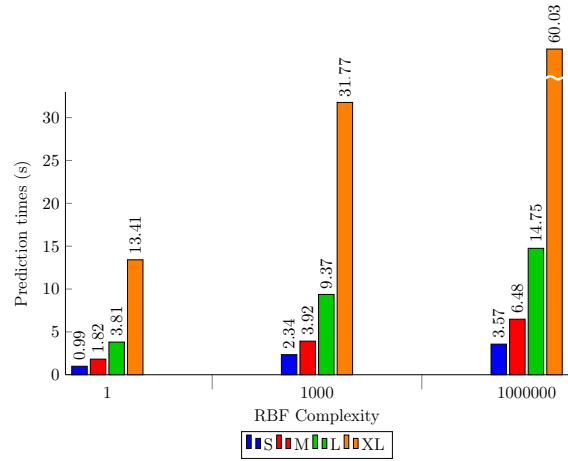
(c) Training times with unscaled features



(d) Training times with scaled features



(e) Prediction times with unscaled features



(f) Prediction times with scaled features

Figure 4.3.: Support Vector Machines

Confusion matrices show how each of the samples from the data set are classified for a given model. As a matter of illustration, a side by side comparison for the worst-performing vs. the best-performing model, with respect to accuracy, is given in Fig 4.4, 4.5 and 4.6. The complete set of confusion matrices, corresponding to every trained model, can be found at the development notebook for this work [Cue17].

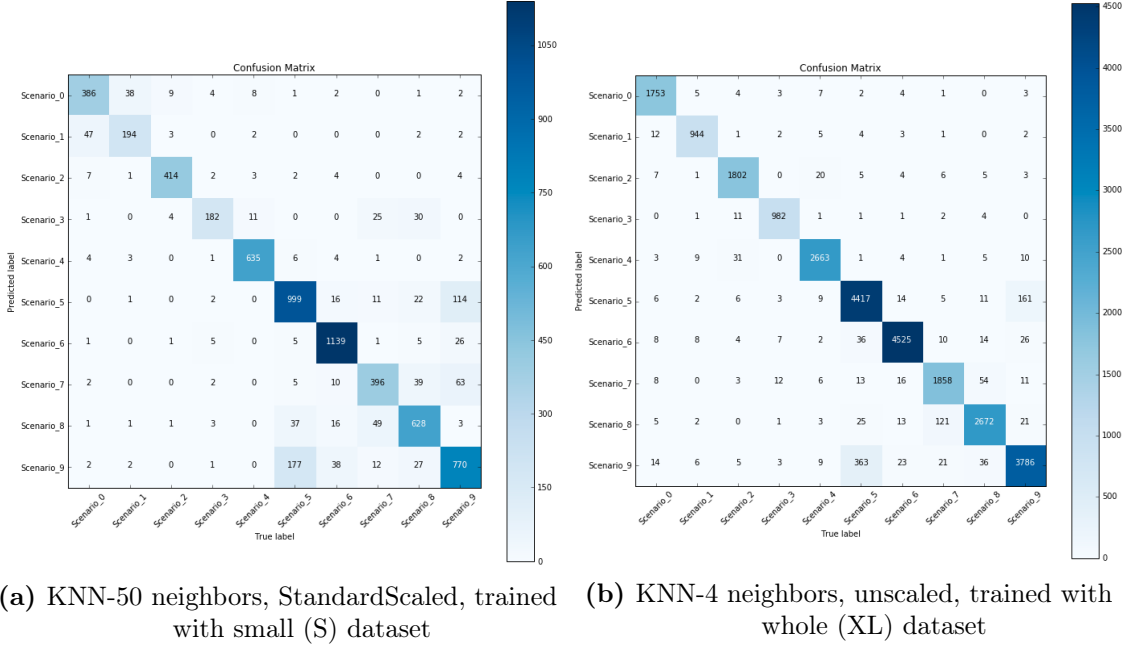


Figure 4.4.: Confusion Matrixes for K-nearest Neighbor Models

It is also important to notice the clear difference on the number of samples present for each scenario. This is due to the fact that the measurements performed in section 3.4.1 were based on runtime and not on number of generated samples, and the feature extraction algorithm described in section 3.4.2 generates a different number of samples for each scenario, because of its dependence on the *frame events* generation consequent of the channel occupation and interframe time of arrival itself. A different approach for feature extraction based on number of features generated is proposed in chapter 6.

From these matrices it can be clearly seen, on a first glance, that the K-nearest algorithm performs quite well regardless of its configuration, having a more populated diagonal in its confusion matrix in comparison with the other two analyzed models. Additionally, just a small improvement in the classification accuracy is seen as the complexity and dataset size increases for this model. In general, it is safe to assert that the classification performs generally well for most of the scenarios except from scenario 5, where a higher number of false positives as well as false negatives is seen for all models, being missclassified as scenario 9. This indicates that the extracted features for these specific scenarios have a noticeable correlation, and serve as an invitation to determine a feature that confidently sets a difference between them.

Based on this analysis, one model of each kind is selected to be implemented on realtime samples. The selected models are those who have the best trade-off between accuracy and

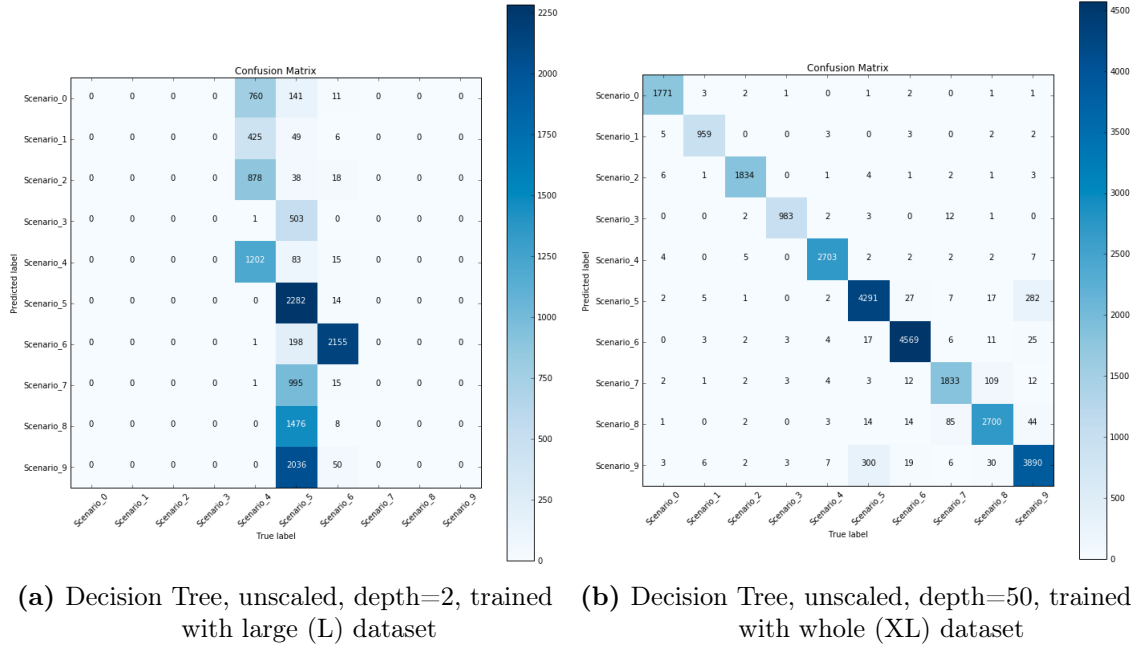


Figure 4.5.: Confusion Matrixes for Decision Tree Models



Figure 4.6.: Confusion Matrixes for SVM Models

prediction time. The selected models are:

- K-nearest neighbors with 4 neighbors, trained with XL unscaled dataset.
- Decision tree with a tree depth=50, trained with XL scaled dataset using a standard scaler.

- SVM with an RBF complexity of $1e6$, trained with S scaled dataset using a standard scaler.

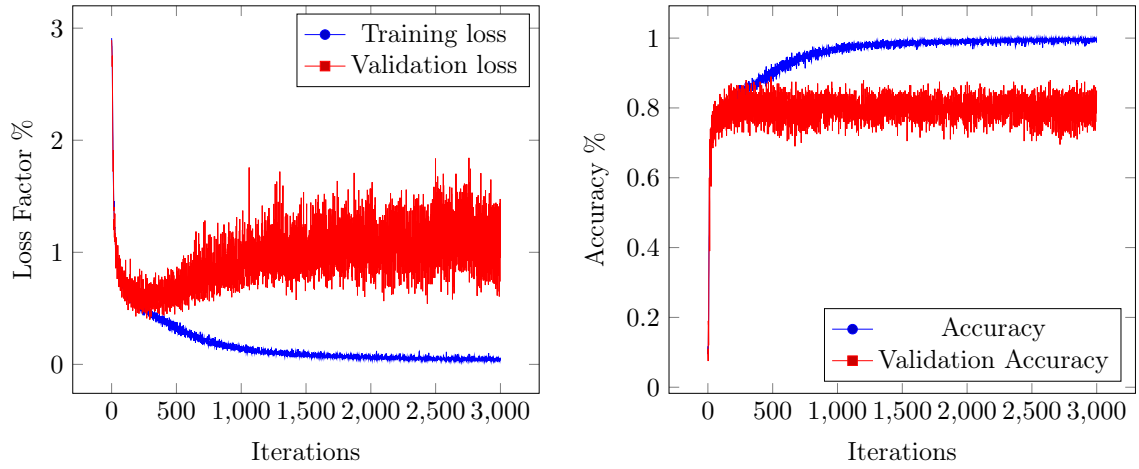


Figure 4.7.: Metrics for Adadelat Optimizer

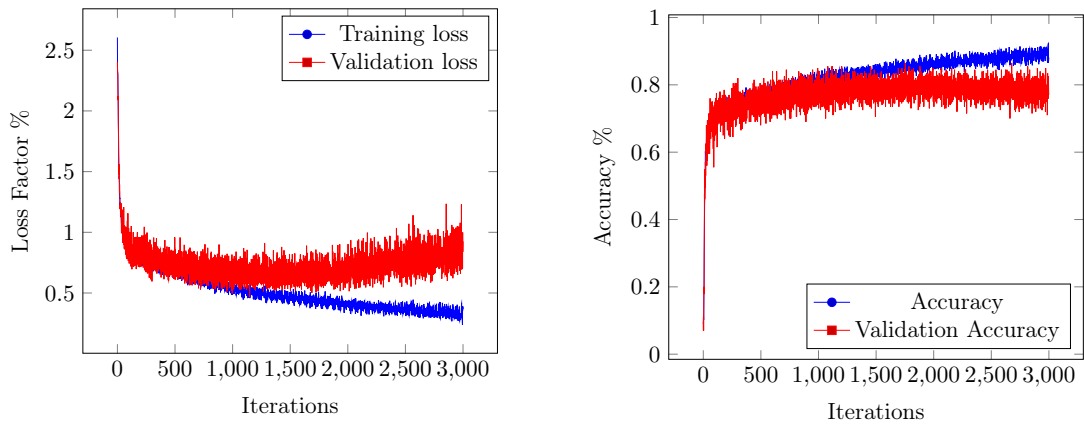


Figure 4.8.: Metrics for Adamax Optimizer

4.3. Dyspan setup comparison

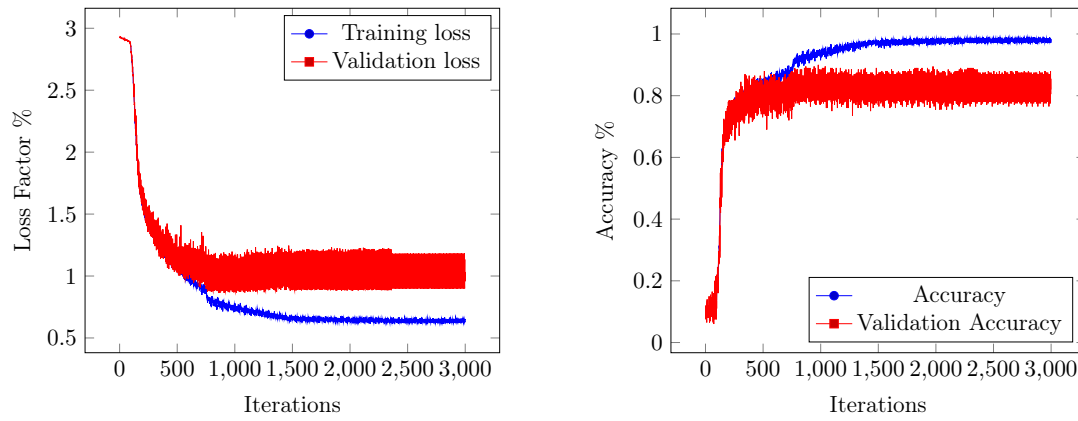


Figure 4.9.: Metrics for Steepest Gradient descent Optimizer

5. Live implementation in GNURadio

6. Conclusion

So you made it! This is the last part of your thesis. Tell everyone what happened. You did something... and you could show that ... followed.

In the end make a personal statement. Why would one consider this thesis to be useful?

A pattern exists.

We cannot pin it down mathematically.

A data set.

Future work

reduce sample rate: modify filters to make it work -> less data saved to disk -> longer observations base feature extraction on sample number, not on time Feature extraction that separates 5 and 9

A. Abbreviations

AI	Artificial Intelligence
BW	Bandwidth
CEL	Communications Engineering Lab
CGRAN	The Comprehensive GNURadio Archive Network
ComSoc	IEEE Communications Society
ConvNet	Convolutional Neural Networks
CR	Cognitive Radio
DL	Deep Learning
DySpan	Dynamic Spectrum Access Networks
EMC	IEEE Electromagnetic Compatibility Society
FFT	Fast Fourier Transform
FCC	Federal Communications Commission (USA)
GRC	GNU Radio Companion
GUI	Graphical User Interface
LDPC	Low-Density Parity-Check
ITU	International Telecommunications Union
IEEE	Institute of Electrical and Electronics Engineers
KIT	Karlsruhe Institute of Technology
ML	Machine Learning
OFCOM	U.K. Office of Communications
OFDM	Orthogonal frequency division multiplexing
OOT	out of tree
PDU	Protocol data unit
pip	Pip Installs Packages

PSK	Phase Shift Keying
PU	Primary User
PyBOMBS	Python Build Overlay Managed Bundled System
RBF	Radial Basis Function
RF	Radio Frequency
SCC41	Standards Coordinating Committee 41
SDR	Software-Defined Radio
SNR	Signal-to-Noise-Ratio
SU	Secondary User
SVM	Support Vector Machines
SWIG	Simplified Wrapper and Interface Generator
UHD	Universal Software Radio Pheripheral (USRP) Hardware Driver TM
USRP	Universal Software Radio Pheripheral

Bibliography

- [AMMIL10] Yaser S. Abu-Mostafa, Malik Magdon-Ismael, and Hsuan-Tien Lin. *Learning From Data*, volume 21. AMLbook.com, 2010.
- [And] Martin Andrews. Nvidia (8.0) installation for TensorFlow on Fedora 25.
- [B21] USRP B210 USB Software Defined Radio (SDR) - Ettus Research.
- [BL07] James Bennett and Stan Lanning. The Netflix Prize. *KDD Cup and Workshop*, pages 3–6, 2007.
- [BM01] Indranil Bose and Radha K. Mahapatra. Business data mining - A machine learning perspective. *Information and Management*, 39(3):211–225, dec 2001.
- [Bro91] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1-3):139–159, 1991.
- [CGR] The Comprehensive GNU Radio Archive Network.
- [Cha] Dyspan Challenge. IEEE DySPAN | IEEE DySPAN.
- [CNT] The Microsoft Cognitive Toolkit | Microsoft Docs.
- [Com] Comsoc. ComSoc Conference Search | IEEE Communications Society.
- [Cue17] Nicolas Cuervo. Github, Cognitive Radio ML. 2017.
- [Fu17] Michael C. Fu. Alphago and Monte Carlo tree search: The simulation optimization perspective. *Proceedings - Winter Simulation Conference*, pages 659–670, 2017.
- [GAMS] Andrea Goldsmith, Syed Ali Jafar, Ivana Mari, and Sudhir Srinivasa. Breaking Spectrum Gridlock with Cognitive Radios : An Information Theoretic Perspective.
- [gnu] GNU Radio - Supported Hardware.
- [GNU16] GNURadio. GNU Radio, 2016.
- [Goo17] Google Inc. Machine Learning – Explore – Google Trends, 2017.
- [Haw] Jeff Hawkins. What Intelligente Machines Need to Learn From the Neocortex.
- [Hay05] Simon Haykin. Cognitive Radio : Brain-Empowered. 23(2):201–220, 2005.
- [HB04] Hawkins Jeff and Sandra Blakeslee. *On Intelligence*. Henry Holt and Co, 2004.

- [Hoe63] Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30, mar 1963.
- [Hsu99] Feng Hsiung Hsu. IBM’s Deep Blue chess grandmaster chips. *IEEE Micro*, 19(2):70–81, 1999.
- [IEE15] IEEE DySPAN. IEEE DySPAN Standards Committee (DySPAN-SC), 2015.
- [IPy] Jupyter and the future of IPython — IPython.
- [Jon05] Friedrich K Jondral. Software-Defined Radio—Basics and Evolution to Cognitive Radio. *EURASIP Journal on Wireless Communications and Networking*, 3:275–283, 2005.
- [Jup] Project Jupyter | Home.
- [KER] Keras Documentation.
- [Kon01] Igor Kononenko. Machine learning for medical diagnosis: History, state of the art and perspective. *Artificial Intelligence in Medicine*, 23(1):89–109, aug 2001.
- [KWS⁺15] Ankit Kaushik, Felix Wunsch, Andrej Sagainov, Nicolas Cuervo, Johannes Demel, Sebastian Koslowski, J Holger, and Friedrich Jondral. Spectrum Sharing for 5G Wireless Systems. (1):6–7, 2015.
- [Lin] Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux - Ettus Knowledge Base.
- [LMH⁺17] Alex Lackpour, Sean Mason, Chase Hamilton, David Tigreros, Matthew Giovannucci, Matthew Marcou, Yuqiao Liu, Marko Jacovic, and Kapil R. Dandekar. Design and implementation of the Secondary User-Enhanced Spectrum Sharing (SUESS) radio. In *2017 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pages 1–2. IEEE, mar 2017.
- [MG16] Andreas Mueller and Sarah Guido. *Introduction to Machine Learning with Python*, volume 53. 2016.
- [MM99] Joseph Mitola and Gerald Q. Maguire. Cognitive radio: making software radios more personal. *IEEE Personal Communications*, 6(4):13–18, 1999.
- [N21] USRP N210 Software Defined Radio (SDR) - Ettus Research.
- [OCC16a] Timothy J. O’Shea, Johnathan Corgan, and T. Charles Clancy. Convolutional radio modulation recognition networks. *Communications in Computer and Information Science*, 629:213–226, 2016.
- [OCC16b] Timothy J. O’Shea, Johnathan Corgan, and T. Charles Clancy. Unsupervised Representation Learning of Structured Radio Communication Signals. apr 2016.

- [OEC17] Timothy J. O’Shea, Tugba Erpek, and T. Charles Clancy. Deep Learning Based MIMO Communications. pages 1–9, 2017.
- [OSX] Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on OS X - Ettus Knowledge Base.
- [Oxfa] Oxford. intelligence | Definition of intelligence in English by Oxford Dictionaries.
- [Oxfb] Oxford. learn | Definition of learn in English by Oxford Dictionaries.
- [PSK⁺17] Francisco Paisana, Ahmed Selim, Maicon Kist, Pedro Alvarez, Justin Talon, Christian Bluemm, Andre Puschmann, and Luiz Dasilva. Context-Aware Cognitive Radio Using Deep Learning. *IEEE International Symposium on Dynamic Spectrum Access Networks (DYSPAN), Spectrum Challenge*, pages 1–2, 2017.
- [PST⁺17] Stefanos Papadakis, Manolis Surligas, Kostis Triantafyllakis, George Vardakis, Athanasios Gkiolias, Nikolaos Karamolegos, and Antonis Makrogiannakis. An agile OFDM cognitive radio engine. In *2017 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pages 1–2. IEEE, mar 2017.
- [PyB] PyBOMBS – The What, the How and the Why - GNU Radio - GNU Radio.
- [Qt] Qt | Cross-platform software development for embedded & desktop.
- [RPP07] Kyle B. Reed, James Patton, and Michael Peshkin. Replicating human-human physical interaction. *Proceedings - IEEE International Conference on Robotics and Automation*, (April):3615–3620, 2007.
- [SKL] scikit-learn: machine learning in Python — scikit-learn 0.19.1 documentation.
- [SW14] G. Staple and K. Werbach. The end of spectrum scarcity. *IEEE Spectrum*, 41(3):48–52, mar 2014.
- [Ten] TensorFlow.
- [The] Welcome — Theano 1.0.0 documentation.
- [UBX] UBX 10 MHz - 6 GHz Rx/Tx, 40 MHz BW - Ettus Research.
- [Win] Building and Installing the USRP Open Source Toolchain (UHD and GNU Radio) on Windows - Ettus Knowledge Base.
- [WKM⁺17] Felix Wunsch, Sebastian Koslowski, Sebastian Muller, Nicolas Cuervo, and Friedrich K. Jondral. A cognitive overlay system based on FBMC. *2017 IEEE International Symposium on Dynamic Spectrum Access Networks, DySPAN 2017*, pages 6–7, 2017.
- [WN07] Zhiqiang Wu and Bala Natarajan. Interference tolerant agile cognitive radio maximize channel capacity of cognitive radio. In *2007 4th Annual IEEE Consumer Communications and Networking Conference, CCNC 2007*, pages 1027–1031. IEEE, jan 2007.

- [WPR⁺17] Felix Wunsch, Francisco Paisana, Sreeraj Rajendran, Ahmed Selim, Pedro Alvarez, Sebastian Muller, Sebastian Koslowski, Bertold Van den Bergh, and Sofie Pollin. DySPAN Spectrum Challenge: situational awareness and opportunistic spectrum access benchmarked. *IEEE Transactions on Cognitive Communications and Networking*, pages 1–1, 2017.
- [X30] USRP X300 High Performance Software Defined Radio (SDR) - Ettus Research.
- [Yad] Vivek Yadak. Deep learning setup for Ubuntu 16.04: Tensorflow 1.2, keras, opencv3, python3, cuda8 and cudnn5.1.
- [ZMJ16] Alessio Zappone, Bho Matthiesen, and Eduard A. Jorswieck. Energy-efficient MIMO overlay communications for device-To-device and cognitive radio systems. In *2016 IEEE Wireless Communications and Networking Conference Workshops, WCNCW 2016*, pages 115–120. IEEE, apr 2016.