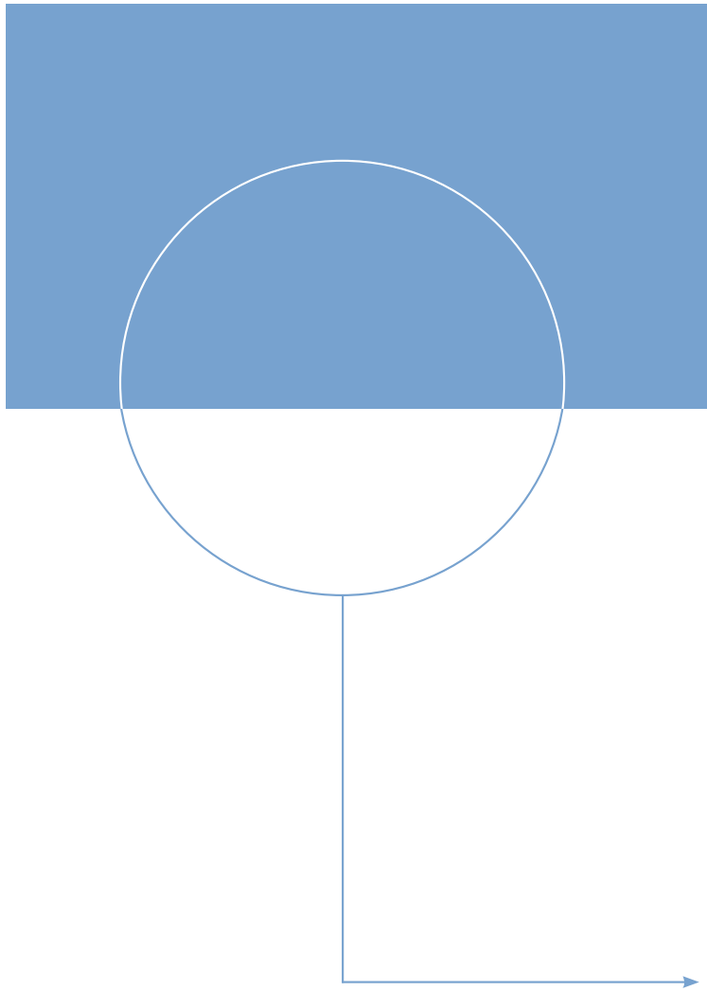

Master's thesis

2022

Piri Babayev

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Computer Science

Piri Babayev

An intelligent decision-making process for hydro scheduling.

June 2022

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

An intelligent decision-making process for hydro scheduling.

Piri Babayev

Artificial Intelligence
Submission date: June 2022
Supervisor: Zhirong Yang

Norwegian University of Science and Technology
Department of Computer Science

Abstract

In Norway hydropower plants are the leading source of electricity production - around 90% of all of the electricity produced [1]. These hydropower plants are built around water reservoirs that generate energy by releasing the accumulated water. The decision when to do so falls upon the hydropower plant operators. As of today, this decision is usually done with hindsight, either based on the previous experiences, or letting the default settings stay as is. There is a strong motivation to optimize this process, to become more cost effective and rely less on other, unsustainable energy sources.

This thesis examines the possibility of utilizing machine learning techniques to predict correct commands given the information about the reservoir volume and inflow, as well as market prices. The historical data provided by 6 Norwegian hydropower energy producers. SINTEF's SHOP (Short-term Hydro Optimization Program) was used to predict the best commands for the given case.

Utilizing TPOT to search a broad space, coupled with preprocessing the dataset using imbalancing data-sampling methods and feature engineering we show that moderate performance can be achieved. Comparing the commands predicted by machine learning pipelines with the default hydropower scheduling setting shows 25% - 100% of non-physical spill cases can be avoided, as well as moderate savings in calculations time and objective value.

Acknowledgements

I would like to thank Zhirong Yang for support and directions in the research and writing of the thesis.

Big thanks to Jiehong Kong and Hans Ivar Skjelbred from SINTEF for helping out in understanding hydropower scheduling systems, and the countless hours spent working experimenting with the results.

I would also like to extend my thanks to my newborn son Elias, who decided that being born two weeks before the submission deadline was a good idea.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Background and Motivation

Currently around 90% of all of the electricity produced in Norway is done via hydropower plants. At the beginning of 2021, the country packed 1681 hydropower plants, producing 136.4 TWh [1]. These hydropower plants usually are placed at the end of watercourses - a string of reservoirs and rivers, streams and canals combining them together into one big system. These reservoirs can be considered as pools of potential energy when not in use, and deciding when to convert this energy into electricity is the main focus of hydropower scheduling optimisation - produce maximum amount of energy based on the water potential in the reservoirs.

At present, the hydropower plant operators manually choose the commands for the hydropower plants before the optimisation models are run. Even though there are a number of methods available to make this choice, operators still prefer to pick commands based on their experience, or simply let the default setting stay put [2]. This manual approach to the scheduling problem limits how effectively a hydropower plant is operated. Making an informed decision is further complicated given the complex web of variables describing the current state.

Depending on the characteristics of the system, data availability, and computational resources, different methods are used for deciding the optimal hydropower scheduling policy. Hydropower scheduling problems can be classified into 4 categories depending on the time interval under consideration [2]:

1. Long-term - 1 to 5 years;
2. Mid-term - 1 to 52 weeks;
3. Short-term - 1 - 7 days;
4. Real-time simulation.

Each scheduling problem has its own mathematical formulation and is solved using different methods. Given these differences of these systems it is not possible to include all the details into one single optimisation model. Long-term hydro scheduling, which is the focus of this thesis, considers several years of the aggregated system description for the scheduling problem [2].

It is possible to simulate the best commands given the state using optimization or simulation tools, like SINTEF's Short-term Hydro Optimization Program (SHOP) [3]. This modelling tool is based on an optimization formulation based on successive linear programming using CPLEX as a solver. At the current state, it has two main problems - it is computationally heavy and hence quite time consuming. The purpose of this thesis is to investigate whether machine learning methodologies can be a good substitute to the current process.

This thesis was part of the larger project called iScheduling project, an intelligent decision-making process for hydro scheduling developed by SINTEF Energy Research. The main goal of the project is to facilitate the decision-making process for hydropower producers by realizing the automatic setup of executive commands before running the optimization tools. The research was conducted with close collaboration with 6 large Norwegian hydropower energy producers - Statkraft, Skagerak Energi, E-CO, BKK, Hydro and TrønderEnergi, as well as SINTEF Energy department.

1.2 Earlier Research

There are a plethora of examples of ML methods used in hydropower operation optimization, albeit the focus has been on a rather narrow selection of opportunities.

[2] gives an overview of the state of machine learning related to hydropower scheduling. Most of the earlier approaches were used in relation to forecasting - streamflow, reservoirs inflow, runoff, etc. [4] documented the usage of Artificial Neural Networks (ANN) which maps the input/output relationship in the operating dams, and have reported moderate success with their approach. [5] shows a successful discharge estimation of the hydropower systems using a linear regression model.

A more recent survey [6] shows that the focus has expanded into scheduling optimisation as well. In [7] a fuzzy neural network was used to find optimal release from a reservoir in order to maximize yearly power generation. This approach has shown high accuracy in prediction power, but it required the flow forecast for up to 12 months. On the other hand, [8] compared the performance of an artificial neural network against the performance of the reinforcement learning model in a long-term multi reservoir systems but their method is problematic in calculation time and suffers from high dimensionality in large-scale systems.

As it can be seen from the overviews, the usage of ML methodologies in regards to hydropower the attention was mainly focused on forecasting inflow and streamflow

or simulations. Hydropower scheduling optimization is often overlooked as an area where ML methods could be applied altogether.

1.3 Research questions

While the goal of the thesis is to research the possibility of using machine learning methodologies in hydropower plant scheduling, in itself, this is a rather broad field, so it is helpful to narrow down to scope. The following three research questions are meant to do exactly that. Throughout the thesis these three questions have assumed the role of the ”red thread” that would direct the research in one direction or another:

Research Question 1: Are there suitable feature engineering methods that could prepare the datasets for machine learning models?

The data in question is a rather big collection of mutually dependent variables, and previous attempts to find models that generalize well have been futile. It is important to find means of improving the quality of the data before ML methodologies are tested. While popular statistical approaches to data preprocessing exist and are easy to deploy, these methods do not incorporate any domain-specific knowledge. This research question aims at finding problem-tailored approach to the feature engineering, and its use in practice.

Research Question 2: Can the issues with imbalanced datasets be alleviated?

The complex datasets at hand have another problem - a lot of them are imbalanced. The imbalance can be described as an issue of class distribution among the examples, e.g. when the number of examples for one of the classes is substantially higher than examples for other classes.

Having a low number of examples for these extreme cases presents a number of problems for classifiers, often leading to overfitting, and worse predictions accuracy for under-represented examples. This issue has its roots in the problem domain as some specific command settings are simply rarely used in practice, since the specific conditions are rather an uncommon occurrence. Finding out approaches that can help mitigate these problems is another cornerstone of this thesis.

Research Question 3: Can Automatic Machine Learning improve modeling and inference?

As popularity of machine learning grows so does the number of available models, each with their own set of strength and weaknesses, and parameters. Attempts prior to this thesis to find good classifiers have been futile, repeatedly resulting in poorly-performing pipelines with accuracy below the random guess. Automatic Machine Learning (AutoML) assists with this problem by automating the process of searching and optimizing machine learning models for the task at hand. There are a

number of available solutions, each approaching the search for the best pipeline in a different fashion, encompassing different libraries and methods.

This also goes hand in hand with one of the goals set forth by industrial partners who are looking for an easy way to construct and set up machine learning models for testing and use in practice. Luckily that is exactly what AutoML is good for! The only question then remains whether it can indeed improve the prediction accuracy of the models for the problem at hand.

1.4 Thesis outline

Since the primary objective of this thesis is to find whether machine learning pipelines can fit to handle the objective set above, the structure will start by discussing the datasets and their nature, then move onto the research approach chapter. Here, the discussion follows the structure of the machine learning pipelines - it first examines the data preprocessing, then the machine learning models and finally the parameter fine-tuning.

Then, the Findings chapter presents the results of this thesis, and gives a comparison between different preprocessing methods and parameter tuning. Finally, the Conclusions and Future Work chapter outlines a few approaches that could be taken into account when working on research questions similar to this thesis.

Chapter 2

Data

A must first step in working with machine learning is examining the data. The importance of this step cannot be overstated since it is the root of many issues and fixes. This analysis of the dataset should reveal details crucial for preprocessing, feature engineering and machine learning models.

2.1 Data Overview

The datasets in question were generated for this project are based on static historical data. Each dataset belongs to one of the watercourses located in Norway. In order to honor the privacy of the companies behind these watercourses, their names have been replaced.

All present features of the datasets can be divided into following categories:

1. Hourly electricity prices;
2. Hourly creek inflow volume;
3. Initial water volume in the reservoirs;
4. End water volume in the reservoirs;
5. Some of the datasets also have a few unique, time-dependent operating constraints features that are watercourse-specific.

Every dataset corresponds to a specific watercourse, which consists of several water reservoirs. Hence, the total number of inflow and initial water volume would vary depending on the number of the reservoirs present. In addition to this, the number of market prices, as well as creek inflow volume columns would depend on the chosen time horizon. The time horizon indicates over which interval of time the historical data has been used to generate the output command.

Table 2.1 below gives the overview of the datasets that have been worked with in this project.

	Years	Cases	Time Horizon	Columns	Reservoirs	Commands	Add. Columns
Partner 1	1	359	168	508	2	1	4
Partner 2	3	1082	72/168/336	440/1016/2024	4	2	0
Partner 3	8	2916	168	854	4	1	0
Partner 4	1	326	240	3866	13	1	0
Partner 5	8	3001	210	634	2	1	3
Partner 6	3	1094	72	522	6	1	6

Table 2.1: Datasets overview.

Two things must be noted. First, because the data is historical, it was possible to generate datasets with varied time horizons. The tests were run separately on all of them, and compared to find whether there was any significant advantage to using shorter or longer time horizon intervals, and it is discussed later in the Various Time Horizon subchapter. Compared to the similar research [6], the amount of available examples is indeed rather low, especially for the datasets with higher feature count.

Apropos of, an additional note must be made when reviewing the number of features in the datasets. The curse of dimensionality is present in machine learning as well, which refers to complications that arise due to a large number of features. The higher is the feature count, the more difficult it is for the classifiers to find relevant features and hence, to generalize well [9]. Approaches taken to mitigate this problem are discussed later in the Data transformation subchapter.

	Command 1			**Command 2**		
Dataset	**Class**	**Distribution**	**Total Count**	**Class**	**Percentage**	**Total Count**
Partner 1	Class 0	61.28%	220	NA		
	Class 1	38.72%	139			
Partner 2 72	Class 0	62.12%	679	Class 0	75.84%	818
	Class 1	37.88%	414	Class 1	25.16%	275
Partner 2 168	Class 0	8.45%	92	Class 0	66.21%	721
	Class 1	91.55%	997	Class 1	33.79%	368
Partner 2 336	Class 0	15.34%	166	Class 0	64.97%	703
	Class 1	84.66%	916	Class 1	35.03%	379
Partner 3	Class 0	60.08%	1752	NA		
	Class 1	39.92%	1164			
Partner 4	Class 0	32.21%	105	NA		
	Class 1	67.79%	221			
Partner 5	Class 0	50.68%	1521	NA		
	Class 1	30.22%	907			
	Class 2	19.09%	573			
Partner 6	Class 0	52.66%	535	NA		
	Class 1	31.10%	316			
	Class 2	16.24%	165			

Table 2.2: Overview of the dataset's class representation distribution.

Dataset	Command 1	Command 2
Partner 1	Overflow_mip_flag	NA
Partner 2	Mip_flag	Overflow_cost
Partner 3	Overflow_mip_flag	NA
Partner 4	Overflow_mip_flag	NA
Partner 5	PQ_points	NA
Partner 6	Mip_gap	NA

Table 2.3: Overview of the commands.

Table 2.2 gives an overview of the dataset's class representation, showing the percentage of input rows belonging to a given class of the command, as well as the total number of input rows per class.

One last note must be made regarding the commands themselves - they differ from company to company, and represent different actions in the system. Table 2.3 below gives a short summary about the commands per watercourse.

An in-depth description of these commands is beyond the scope of this thesis, but a brief explanation is given for completeness:

1. Overflow_mip_flag: indicates whether binary variables should be used in the mathematical formulations of overflow from the reservoirs.

2. Mip_flag: indicates whether binary variables should be used in the mathematical formulations of the start-up decision of generating/pumping units in each plant and each time step.

3. Overflow_cost: the cost of the overflow, which is included in the objective function of the optimization problem.

4. PQ_points: the number of breakpoints on the unit Production/Discharge (PQ) curve.

5. Mip_gap: the gap dictating the classification of PQ points on the curve.

It must be noted that while the data is historical, the correct value for these commands were arrived at using SINTEF's SHOP program [3]. After simulating the conditions present in the case, the program presents values that are optimal for maximum energy generation. It must be noted that this process is rather time-consuming, hence limiting its application in decision-making in practice.

2.2 Class imbalance

The overview above in Table 2.2 indicates that a few of the datasets have an imbalancing issue. Any dataset in which the representation of observations of the given classes has an uneven distribution by a substantial margin [10] [11] can be classified as imbalanced. In these datasets, the underrepresented class is called the minority class, while the others are called the majority class.

As one might suspect, these datasets are plagued by a number of problems. For the start, machine learning models are likely to overfit, representing only the majority class well. As a consequence, traditional approaches to measuring the performance of the models might be misleading. The problem with picking a correct evaluation metric is discussed later in the Evaluation Metrics subchapter.

Additionally, given that there are fewer representations of the minority class in such a high feature count datasets, there simply might not be enough examples for the classifiers to learn to correctly predict them. There are a number of approaches that can be taken to counteract these issues, and the methods taken are discussed in the Approaches to combat Imbalanced datasets subchapter.

2.3 A closer examination of the datasets

A brief look into the contents of the datasets could be helpful when deciding on preprocessing methods and its parameters. Below in Figure 2.1 an overview of a test system is presented, spanning 1 year, from January 2017 to June 2018, consisting of 9 reservoirs. This simulation was run using SINTEF's SHOP [3].

The market price chart indicates the prices of electricity at the given hour. The spikes and dips often coincide with the inflow to the reservoirs and the amount of volume present in the reservoirs at the time, which is not surprising given that a big portion of the Norwegian electricity is generated using hydropower.

The shape of the inflow graph indicates high spikes in the inflow around spring, when the accumulated winter snow starts melting, as well as in autumn, it being a relatively rainy period in Norway, with a relatively "dry" season in winter, due to sub-zero temperatures, where next to no inflow is present. Additionally, there are

”stochastic” changes in the data, which are attributed to sudden weather changes. It is important to point out that these sudden changes could be crucial in the scheduling decision making process, as these changes happen in the course of just a few hours, and have strong effects on the scheduling optimisation.

The last graph shows the total volume of the water in the reservoirs. The seasonal changes are quite apparent here as well.

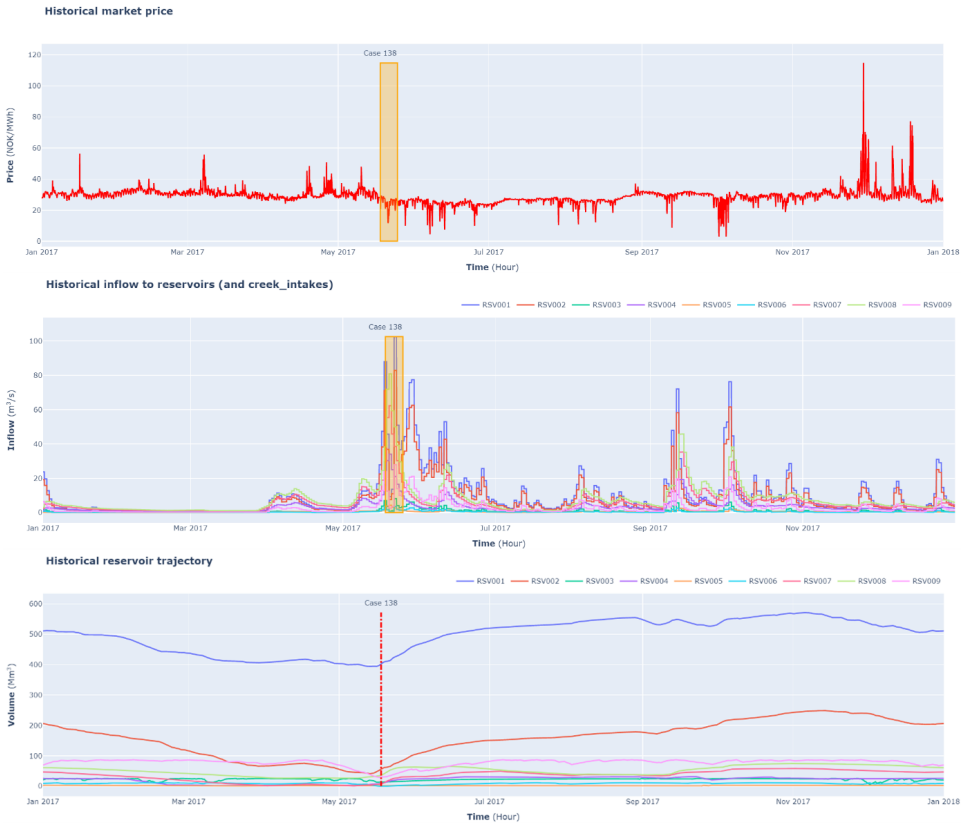

Figure 2.1: Dataset content overview of a test system.

Chapter 3

Research Method

This chapter examines the approaches that were taken in this project. It starts off by examining how the problem description determined the machine learning models to use; how the characteristics of the datasets' content informed the pre-processing methods applied; and then the general work process that was adhered to to achieve the results presented in the next chapter.

3.1 Machine Learning

This section aims at defining and examining the approaches in machine learning related to the project. It briefly discusses the important terms and details that one needs to be aware of.

3.1.1 Supervised Learning

Given the project description, supervised learning was a natural approach to tackling the problem at hand. Supervised learning refers to the subclass of machine learning in which the machine learning model, or classifiers, learn to generalize on the posed problem given the example set. These examples can be divided into the input and output sets, with the former being a "question", and latter being the correct "answer" [9].

In the case of this project, the input are the features, and the output, also known as classes being the commands. This problem can also be categorized as a classification problem - the end goal of the models is to correctly identify to which class the input set belongs to. It is usually done by presenting the classifier with part of the dataset consisting of both input and output sets. This stage is known as training or fitting. Afterwards, the model is asked to classify the previously unseen input set, and the correctness of the model is then evaluated, in the step known as testing. The process of splitting the dataset in two is described in Splitting.

3.1.2 Ensemble Methods - Stacking

Ensemble methods followed "Two heads are better than one" advice quite literally - why use only one machine learning model for the problem, when one could use two? Ensemble methods work in this vein by training and using more than one model for the problem. Once each model had their say, this result is aggregated in one form or another [12].

Stacking does so by adding another layer on top of the existing models, by introducing another one often called *meta-model*. This model is trained to combine the output of the previous layer and produce the final prediction for the input example. This approach has the strength of utilizing several machine learning algorithms, which might generalize better for one class or another [12] [9].

Additions to the complexity of the problem solving further complicates the interpretability of the results, making it hard to derive useful information to advance the work in the field. Another problem with this approach is the diversification of the models in stacking. The combination of high accuracy models does not necessarily lead to increase of prediction power of the meta model, as all models together could be subject to the same bias that could result in over-fitting [13].

3.1.3 Overfitting

Overfitting is a common issue in machine learning field. It refers to cases where on top of learning to generalize over over the training examples, the model also learns the noise in the dataset. This way, its function fit the examples too well, resulting in high accuracy for the given examples, but as a result performs poorly on previously unseen dataset. [9]

This issue can also occur when the model learns to generalize over imbalanced datasets, learning to represent one class, usually the *majority* one better than the other [14]. This could lead to erroneous conclusions about model's ability to generalize well beyond the seen examples. That is why it is important to notice this issues in advance, and make corrections in time.

3.1.4 Evaluation Metrics

In order to evaluate and compare trained machine learning models a good evaluation metric is required. Accuracy is the widely-used option which simply indicates the percentage of correctly predicted labels [9]. In practice this metric falls short on a few crucial points - importantly it gives no information about how well the model performed on the per class basis. As an example, an 84% accuracy score for the model trained to predict Command 1 of the Partner 2 dataset, gives no information whatsoever of whether the said model performs well for both classes, or simply

overfits to represent Class 1 only. This is particularly important for evaluating the performance on the imbalanced datasets, as pointed out in [11].

Additionally, as discussed earlier, the evaluation metric plays an important role in comparing two models, or different parameter sets of the same model, especially when it comes to fitness function in the TPOT, discussed below in TPOT. Balanced Accuracy is a metric suggested both by [11] and in the documentation of [14] to counteract these issues. It can be calculated by finding the arithmetic average of true positive rates (also known as *Sensitivity*) and true negative rates (also known as *Specificity*). *Sensitivity* indicates the percentage of the true positives that have been identified by the model. On the other hand, *specificity* indicates the percentage of the true negatives. Intuitively then, they can be calculated by:

$$Sn = \frac{TP}{TP + FN}$$

$$Sp = \frac{TN}{TN + FP}$$

Where *Sn* is *Sensitivity*, *Sp* is *Specificity*, *TP* is true positives, *FN* is false negatives, *TN* is true negatives and *FP* is false positives. Balanced Accuracy then can simply be calculated as average of these values:

$$BA = \frac{Sn + Sp}{2}$$

This definition can be extended to accommodate multi-class problems as shown in [15], by calculating sensitivity and specificity per class, and averaging over number of classes present:

$$BA_{mc} = \frac{1}{n} \sum_{i=1}^{n} \frac{Sn + Sp}{2}$$

Where n is the number of classes present. In a dataset where the class distribution is perfectly equal, balanced accuracy score equals accuracy [16]. It must be noted that there is another definition for balanced accuracy score in the literature, which is the average of recall per class [17]. To keep the confusion to the minimum this project has used only the former definition of the balanced accuracy, and not the latter.

3.1.5 Parameter Tuning

Usually, after decent results have been achieved, it is customary to further improve the performance of the model by tweaking its parameters. This process of finding out the right set of hyperparameters in order to maximize the performance of a

classifier is a process known as *parameter tuning* and in some literature *hyperparameter optimization*[18]. This process is not limited to machine learning parameters only, but can also be applied to preprocessing algorithms as well, as discussed in Optimizing Feature Engineering Parameters and Effects of Imbalanced learn subchapters.

This process can result in significant changes for the predictive power of the models, and the output of preprocessing algorithms, but often require knowledge of both the dataset and the model. This creates a problem not only in choosing the right model, but also the right parameters for it. The parameters can indeed be either changed dynamically throughout the learning process, or driven by another search algorithm like Optuna [19]. The question whether the correct machine learning model was picked in the first place is still remains unanswered though, resulting in "chicken or egg" dilemma.

The approach in this paper was utilize the tool that combines both *architecture search* [20] and *hyperparameter optimization* by automating the search process. The discussion of this topic continues below in section 3.3.

3.2 Preprocessing

Preprocessing refers to one of the first steps in the machine learning approaches where data is analyzed and prepared for the classifier [21]. As it points out, low-quality data will lead to a low-quality model performance. There are many approaches one can take in preprocessing the data - a summary of those approaches can be found in [21]. Many of these, statistically-based approaches are integrated into the TPOT algorithm as part of the search. As such, the discussion here is kept only to the "in-house" approaches that were used in the project.

3.2.1 Data transformation

Refers to the processes where data is transformed through feature aggregation, feature engineering, generalization etc. Each of these are applied to the dataset independently, with various effects on the performance of the models in the process. Feature engineering refers to the transformation that is applied to the dataset using the domain knowledge. Examining the data, and understanding its features could unveil new ways of preparing the dataset.

As such, the nature of datasets hints at an instinctive way one could approach feature engineering - instead of keeping all features related to the time-related input columns, one could find the mean and/or standard deviation over n number of hours. This way, both market prices and water inflow volume columns can be shrunk significantly, resulting in less input features. In the Findings chapter a closer look is taken to the performance of this approach, as well as the proposed values for n.

3.2.2 Splitting

Once the data is preprocessed, it needs to be splitted into two sets - one for training the model and one for testing (usually between 20% to 33% of the entire data). These sets include mutually exclusive parts of the dataset to ensure that the evaluation happens only on the previously unseen data, in order to give a better estimation to its performance. The process of splitting itself is stochastic - the algorithm randomly picks input rows when sorting them into training and testing datasets. Thus, output of the splitting for the same dataset is quite likely to always be different.

The training dataset should be a good representation of the main data - which, given the imbalanced datasets and the random nature of the splitting procedure might not always be the case. To counter that, the process is stratified - e.g. the splitting ensures that the distribution of the classes in the two sub-datasets is roughly the same.

3.2.3 Approaches to combat Imbalanced datasets

As discussed earlier, the imbalance of the dataset can be a source of many issues. The easiest solution would be to drop some of the input rows for the majority class(es) to ensure that every class is represented equally. This is rarely a desirable option, since trained models then do not learn from all of the available data. It could also result in models completely missing some of the edge cases, leading to worse prediction capacity. Additionally, some of the datasets as seen in the Data chapter would be losing a massive chunk of available training data altogether.

Luckily, there are quite a few possible options one could explore to alleviate this problem. [11] divides potential solutions into 4 groups:

1. Algorithm level - these approaches adapt the existing algorithms to skew their learning towards the minority class. This usually requires an intimate knowledge of both the problem domain and the algorithm at hand;

2. Data-level or preprocessing - these methods are applied on the training dataset, to improve its quality before models are trained;

3. Cost-sensitive - these approaches integrate both algorithm and data-level approaches together;

4. Ensemble-based - as the name hints, this is a combination between an ensemble learning algorithm and one of the techniques above;

As [link] gives a short overview about the application of these approaches in different domains. For this project, the focus was on data-level and ensemble-based approaches.

For the data-level methods in this project the imbalanded-learn library [22] has been used. It gives access to a number of popular sampling methods. Three of them have been used in this project, all belonging to sampling methods of the data-level approach [10]:

1. Random Under Sampler - reduces the number of the over represented classes by randomly removing them, and generates new samples for under represented ones using the original set. This approach will work well in cases where there is abundant data, or the imbalance is not too skewed towards one of the classes. In the datasets where the total number of input rows is low this approach will result in a loss of information, leading to the machine learning models generalizing worse.

2. Random Over Sampler - generates new samples by randomly sampling with replacement the under represented classes. The term replacement merely indicates that the chosen input is still available to be sampled again even after it was picked by the algorithm. These approaches will be effective for the models that iteratively approximate to the desired functions (like neural networks) and seek good splits (like decision trees).

3. Synthetic Minority Over-sampling Technique (SMOTE) - is a more refined hybrid approach to over-sampling which combines both under- and over-sampling. It uses k-nearest neighbors to create new samples using interpolation. This is a far more sophisticated approach than the ones described earlier, but the quality of the synthetic data can vary greatly from dataset to dataset.

As mentioned earlier, all three of these approaches are only used on the training data, leaving the testing data untouched. The classifiers then would learn to generalize on a mix of both synthetic and real data, while the evaluation would strictly be limited to the latter. This way the expected performance of the model on the real world data is likely to be the same.

Because of this approach, the testing set represents the original distribution of the classes as shown in the Splitting subchapter, and hence is still imbalanced. It is vital to take this into account when deciding on the evaluation metric, in order to not misinterpret the performance of the trained classifiers. The balanced accuracy metric alleviates this problem by including the information how well the model has performed for all the present classes (e.g. sensitivity and specificity).

3.3 AutoML

Finding the machine learning model, as well as their hyperparameters, alongside the preprocessing methods and its parameters can be a very time-consuming process. This is where AutoML methodologies step in. They are designed to reduce

the manual effort related to machine learning, accelerate the experiments and deployment [23]. Quite often the output of AutoML is a machine learning pipeline - which are defined as a combination of data processing techniques and machine learning models, essentially creating an automated system that takes in the data, and produces predictions for the cases [24]. Another important aspect of AutoML is hyper parameter optimization of the said machine learning methods. The optimization methods vary from method to method, but have the common goal of optimizing the performance of the models by tweaking their parameters [24].

Application of AutoML are numerous, and interest in this topic in the scientific community has only been rising [23] [24]. Surveys [23], [24] and [20] give a broad overview of existing solutions, as well as their application domain - which vary greatly from image recognition and recommendation systems to natural language processing and object detection etc. .

Having praised AutoML, one also needs to mention main drawbacks with their application. Reproducibility and interpretability of these methods are still rather obscure, owing it to the automated process. It is hard, if not impossible altogether [24] to determine why a particular preprocessing algorithm or machine learning model was chosen in a given pipeline. The completeness of the pipeline are also another issue, as pointed in [20]. There is a plethora of available machine learning methods and approaches to preprocessing, and the question of whether the entirety of these opportunities are used in the AutoML framework.

3.3.1 TPOT

TPOT, the choice of the AutoML algorithm for this project, automates the process of finding machine learning pipelines by utilizing a genetic algorithm to search a vast search space of potential preprocessing tools, machine learning models and their parameters. It does so by adapting the Genetic Algorithm (GA), which fares notably well in optimisation problems [25]. Genetic Algorithm, as the name hints, is inspired by Darwinian evolution. First, a pool of potential solutions, known as population, is randomly created. Each of these solutions is an independent instance representing a machine learning pipeline, e.g. the genetic information. In the context of GA these instances are called individuals. This population then goes through a number of stochastic changes, known as genetic operators. They can broadly can be summarized into two categories:

1. Recombination - the genetic information from two or more individuals is combined to create a new individual;
2. Mutations - changes to the genetic information are made within the individual, where one or more elements are changed.

This results in a new population, which is then tested for fitness. Each individual gets a fitness score, which, in the context of machine learning, is how well it

performs on the given dataset. The best performers, mixed with some random individuals are then taken to the next generation in the step called selection, and the process is repeated until the set conditions are met - usually either until there is no significant performance gain over generations or the total number of generations set by the user is achieved.

The correct evaluation process of the individuals in the selection step is vital for the algorithm to find pipelines that result in best solutions, since that is the value that the GA is trying to optimize. If the chosen evaluation metric does not represent the optimization goal well, the algorithm might arrive at the wrongful conclusions. The outcome of the discussions in the Evaluation Metrics and in Class imbalance subchapters guided the decision to use balanced accuracy as the fitness evaluation score. This ensured that all classes were equally important in the evaluation process. Since the search is driven to maximize the evaluation metric, testing showed that if evaluation metrics that do not take these issues into account are used, the TPOT is likely to overfit for majority class.

Number of the individuals in the population in each generation, the amount of individuals that experience mutation, the number of new individuals created by recombination, number of generations etc. depend on the settings of the GA, some of which change as the algorithm progresses through the generations. The chosen parameters for the TPOT are presented in the Findings chapter.

TPOT is built on top of popular public library known as *scikit-learn* [26] utilising the available regression and classification models. In addition to these traditional methods it uses *Pytorch* library [27] for neural network models, and a more recent gradient boosting tree model, *XGBoost* [28]. At the point of writing, it lacks the support deep-learning models, as well as some more modern models, e.g. CatBoost, LightGBM [20].

On the bright side, in addition to finding the best models for the task, in literature referred to as *architecture optimization* [20], TPOT also includes hyper parameter optimization for the said models. This deepens the search space, and contributes to the completeness of the AutoML pipeline.

3.4 IDUN

One of the main drawbacks with any AutoML algorithms is the demanding computational power and execution time - the default settings of TPOT consider 100 generations of 100 individuals each, which ends up evaluating roughly 10 000 machine learning pipelines, which in turn have a number of preprocessing algorithms and might consist of several classifiers. To find good solutions it is recommended to run TPOT over a longer period of time [29], which requires time and a lot of computational power.

For this project, NTNU's IDUN, a high performance computer group, was used [30]. It was possible to run several instances of TPOT at the same time, shortening the computation time.

3.5 Reproducibility

Due to the sensitive nature of the datasets' contents, they are not available publicly on the git channel H which poses some difficulty if datasets of similar structure are not available at hand. Nevertheless, the implementations of the approaches discussed here and in the Findings chapter are available in the repository, along with a short guide on how to get started.

It is important to note the results obtained by running the TPOT algorithm can vary by a wide margin, even when run for the same dataset. There are a multitude of available ML models, preprocessing tools and their parameters, and given the randomness within the recombination and mutation stages in the GA, there are many possible combinations that would result in wildly different pipelines, but similar scores.

The stochasticity of the algorithm, along with the set parameters, also affects the highest score achievable in a given run. Stochastic nature of GA also has an ever so slight probability of TPOT yielding results below the ones achieved in this project.

One should also take care to apply custom feature engineering preprocessing steps before the dataset is fed to the TPOT, as well as ensure that the splitting of the dataset is equal when comparing different settings of preprocessing step and TPOT. Because of this, one can think of TPOT optimizing the particular datasets - e.g. the split and synthetic data generated by the imbalanced learn algorithms. Since all these processes are stochastic in their nature, they might differ by a wide margin when replicating the results. Hence, it is advisable to save the "newly generated" dataset after the stochastic processes have had their go, in order to properly test the TPOT's suggested pipelines.

Small number of examples in some datasets leads to a problem with perception of model's performance - a single correct/incorrect classification has a big impact on the evaluation score, leading to false assumptions about model's performance. To avoid falling into this bias, the distribution of correctly and falsely predicted commands should be reviewed at all times as well, ensuring that understanding of model's performance is correct.

Chapter 4

Findings

This chapter focuses on the results obtained in this project. It starts by analyzing the findings related to the datasets and their preprocessing parameters, as well as how the variations of the time horizons within the same dataset influences the performances of the classifiers, as well as the effects of imbalanced data sampling methods on the predictive power of the machine learning models. Finally, a closer look is taken to the results produced by TPOT.

4.0.1 Optimizing Feature Engineering Parameters

This subchapter looks at the parameters for the feature engineering part of the preprocessing step. As mentioned earlier in the Preprocessing subchapter, the goal of this step is to reduce the number of features, improving the quality of the data which would lead to better predictive power of the classifiers.

To test this hypothesis two preprocessing parameters in the feature-engineering step will be tested:

1. Number of columns over which the data is aggregated for the time-related features as discussed in Data transformation. This value can be any of the divisibles of the dataset's time horizon;

2. Whether standard deviation for the aggregated column should be included or not;

Given that the number of parameters to test and their potential value create a limited and relatively small search space, simple exhaustive search [31] would be sufficient to test all potential combinations, and is part of the parameter tuning process. The base machine learning model for all the optimisation steps was kept unchanged, with out-of-the-box default settings for Random Forest classifier, which shows good results out of the box for many problems [31]. This way, the main focus

would be on how the data transformation affects the predictive capabilities of this simple classifier.

To minimize the effects of the stochasticity in the training process, for each preprocessing setting the same classifier was fit 100 times, and the average of this batch run was used as the evaluation metric. The main goal here is to empirically study the effects of the proposed approach. Following plots in Figure 4.1 illustrate the effects of the settings' parameters. The x-axis represents the interval over which the aggregation is done, with the orange line representing the cases where standard deviation was included. The left-most value on x-axis indicates no preprocessing included at all, e.g. the data fed to the classifier is as is.

As it is clearly seen from the graphs, all of the datasets performed better after some preprocessing was applied, although not always by a significant amount.

When averaging over a smaller intervals - e.g. 2,3 or 4 hours, feature reduction is minimal, so the issue with high dimensionality still persists, albeit to a smaller degree. On the other extreme, when averaging over higher intervals, the feature reduction is significant. This seems to have an adverse effect which can be explained by aggregation being too destructive, leading to information loss, and consequently resulting in classifiers failing to generalize well. That seems to be the case with Partner 1, Partner 2 and Partner 3. These are also the datasets that show significant improvements with preprocessing turned on.

For the rest of the datasets it is not the case. As pointed earlier in the Data chapter, extreme weather would result in unexpected spikes and dips in the market prices and inflow combinations. When aggregating over larger time intervals these important cases are lost, and with that so does the possibility of the classifiers to recognize them in the training process.

One curious find is that almost all datasets perform better when aggregating over the 2 hour interval. This can be characterized as the middle ground between feature reduction and the degree of destructiveness of the aggregation method. The inclusion of standard deviation either has the adverse effect on the classifiers, or to have barely any effect - and with the aggregation interval set to 2 keeps the total count of features unchanged compared to raw data.

The only deviation in these findings is the Partner 4's dataset - the best preprocessing value is the polar opposite compared to all others. This might be caused by its enormous feature count. With 13 reservoirs this dataset packs a whopping 3 866 input features, represented only by one year of historical data. These proportions could be the reason behind the results.

The overview of the balanced accuracy without any processing, and the best parameter settings found are given in Table 4.1.

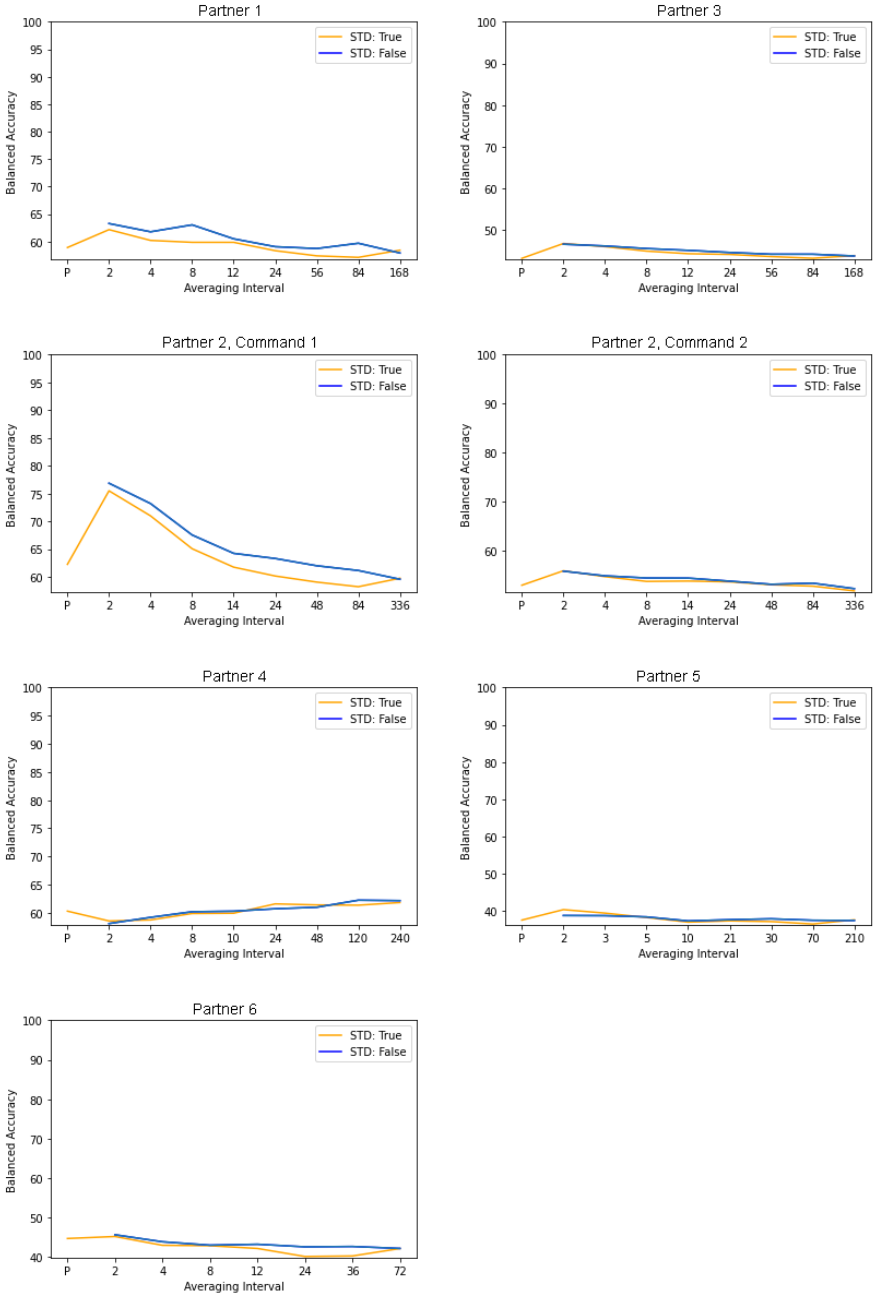

Figure 4.1: The overview of Random Forest classifier with varying averaging intervals of the feature engineering process.

	No preprocessing	Best preprocessing settings		Balanced Accuracy
		Interval	Standard deviation	
Partner 1	58.5%	2	False	63.2%
Partner 2	Command1: 62.2%	2	False	76.8%
	Command2: 52.9%	2	False	55.7%
Partner 3	46.7%	2	True	43.2%
Partner 4	60.3%	210	False	62.2%
Partner 5	37.6%	2	True	40.4%
Partner 6	44.7%	2	False	45.6%

Table 4.1: The overview of Random Forest classifier with varying averaging intervals of the feature engineering process.

4.0.2 Time Horizon

Since the datasets are based on the historical data, it is possible to generate datasets with various time horizons. The dataset with larger time horizons essentially take into account longer time periods. One of the questions posed during the project was how well would a classifier perform given various time horizons for the same dataset.

To examine them, Partner 2's dataset with 72, 168 and 336 -hour time horizons have been used. Because this dataset is the most imbalanced out of the bunch, balanced accuracy is used for comparing these datasets. Additionally, since the time horizons are different, the preprocessing settings discussed earlier can vary for these variations of the datasets. Hence, the same exhaustive search as done above was used here. The comparison is given below in Figure 4.2.

First off, the intervals for feature aggregation chosen are all the same for all of the variations of the same datasets, for both commands. This fits with the observation made earlier in the previous subchapter.

The other significant find is the inverse change of the performance in different commands as the time horizon changes. Classifier performs better in predicting command 1 significantly better as the time horizon increases, while it gets worse for command 2. The overview of the datasets in the Data chapter helps explain

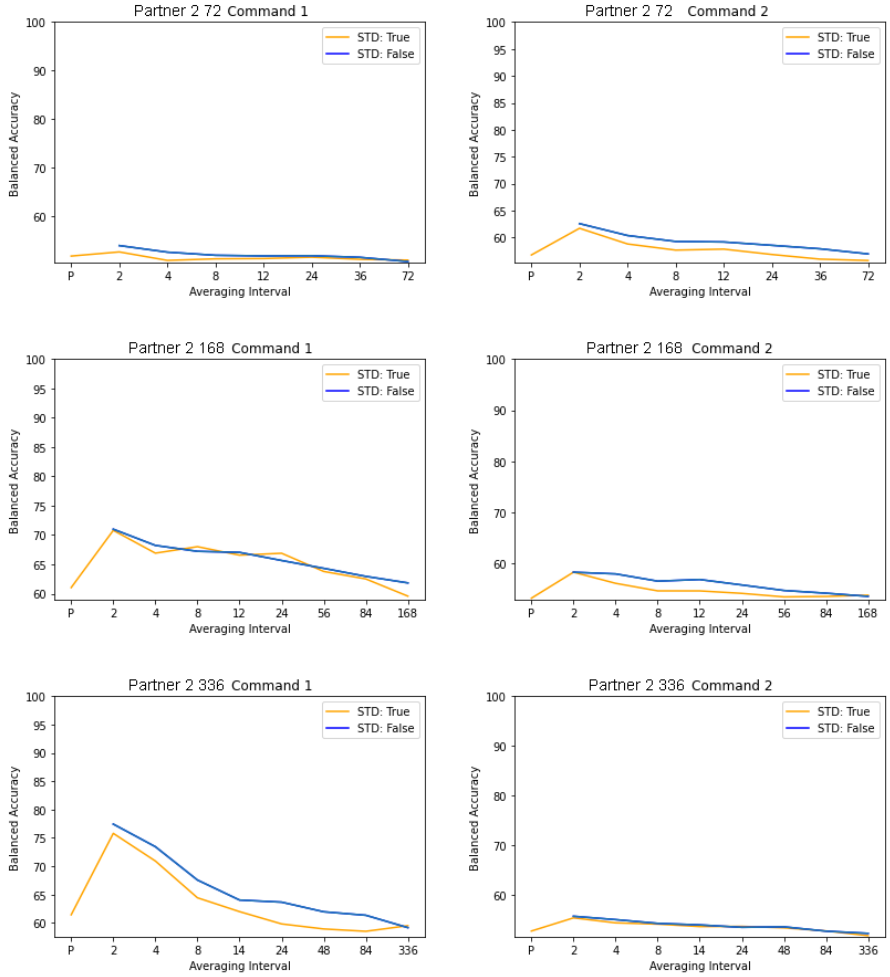

Figure 4.2: Comparison of Random Forest classifier with varying averaging intervals of the feature engineering process for a dataset with various time horizons.

the situation - the distribution of the classes is not the same across the varied time horizons, which explains the inverted performance values of the classifier.

The overview is given below in Table 4.2:

One important takeaway from these tests is that to get the highest predictive accuracy, it would be possible to create datasets with varied time horizons for different commands. These datasets can be the cases for the same watercourses, but treat them as separate problems.

	Accuracy with preprocessing turned off	Best preprocessing settings		Balanced Accuracy
		Interval	Standard deviation	
Partner 2 72	Command 1: 51.8%	2	False	54%
	Command 2: 56.7%	2	False	62.6%
Partner 2 168	Command 1: 61%	2	False	71%
	Command 2: 53.2%	2	False	58.3%
Partner 2 336	Command 1: 61.4%	2	False	77.4%
	Command 2: 52.7%	2	False	55.7%

Table 4.2: Overview of a dataset with different time horizons.

4.0.3 Effects of Imbalanced learn

This subchapter examines the effects of the approaches to the imbalanced datasets, namely how the chosen data-level approaches affect the performance of the classifiers. Since all three data-level algorithms that were discussed in Approaches to combat Imbalanced datasets subchapter use the existing data to either replace or create new rows, its performance may be affected by the preprocessing settings of the data transformation algorithm. The exhaustive search once again is used, testing all three sampling algorithms, with no preprocessing involved, as well as the best preprocessing parameters found in the previous chapters.

First, the analysis for Partner 5 and Partner 6 datasets are presented. Both have 3 classes with a high degree of imbalancing issue, with one of the classes occupying half of the present examples. Then, the effects of sampling algorithms are tested on Partner 3, Partner 4 and Partner 1. These three datasets are plagued less by the imbalancing issue - the examples are divided 7:3 for Partner 4 and 6:3 for the rest. Nevertheless, it would be interesting to see whether alleviating this mild imbalancing issue will help to improve the performance of the classifiers. Lastly, Partner 2 is evaluated separately - varied time horizons affect the performance of the classifier, hence the impact of sampling methods may differ too.

Following in Figure 4.3 are the results from the search. The x-axis represents the sampling methods used, with the orange line representing the cases where preprocessing was not used. The left-most orange point on the x-axis again is the baseline - unprocessed data with no sampling methods applied.

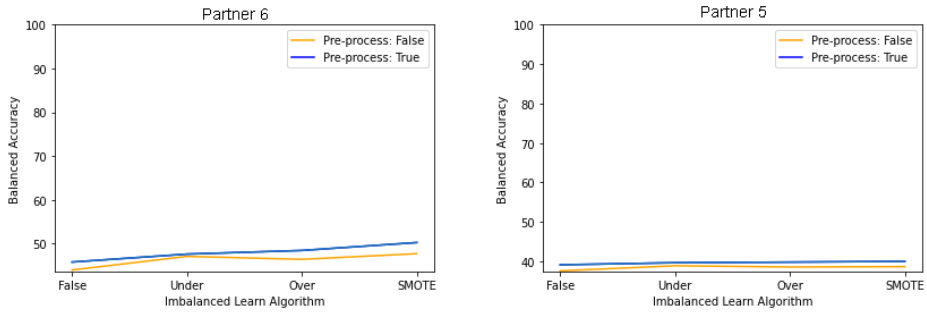

Figure 4.3: Comparison of the effects of the sampling methods on the highly imbalanced datasets.

Both Partner 5 and Partner 6 both perform better when both preprocessing and a sampling method are enabled - gaining roughly 3% for the former and 6% for the latter in performance. The SMOTE algorithm is also a clear-cut winner when it comes to picking the right sampling method for these datasets.

The Table 4.3 below sums up the results from the search for these two datasets.

Next, Partner 3, Partner 1 and Partner 4 datasets. The first two show that the combination of the preprocessing and a sampling method allow the classifier to generalize better. Roughly 5% increase in both cases, with SMOTE again in the lead. The comparison is given below in Figure 4.4.

Partner 4, on the other hand, is the only dataset that shows better results when the preprocessing is turned off and the Under sampling method is applied.

The Table 4.4 sums up the results from the search for these three datasets.

	Pre Process	**None**	**Under**	**Over**	**SMOTE**
Partner 5	False	37.5%	38.8%	38.6%	38.6%
	True	39.1%	39.6%	39.5%	39.9%
Partner 6	False	43.9%	47%	46.4%	47.7%
	True	45.8%	47.6%	48.4%	50.2%

Table 4.3: Overview of data-sampling methods to combat imbalance for highly imbalanced datasets.

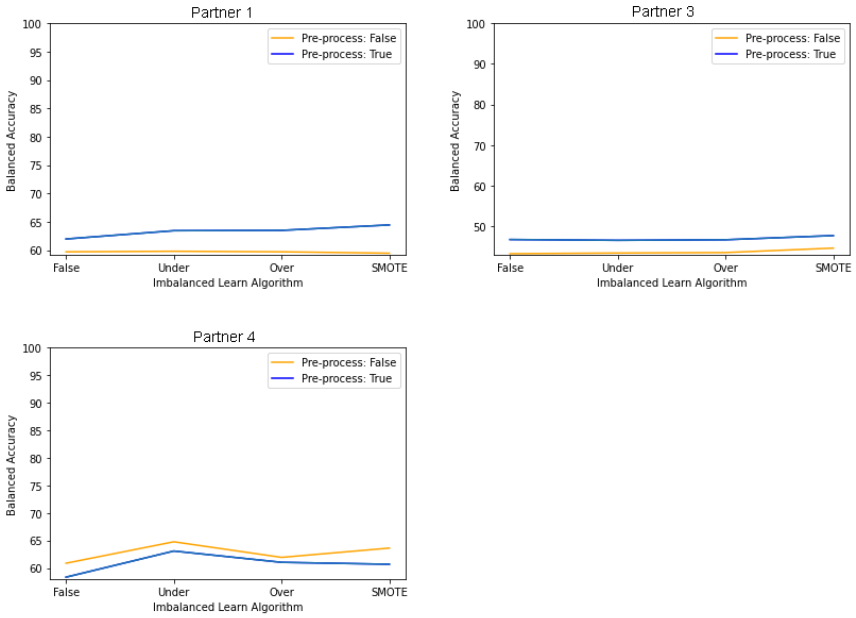

Figure 4.4: Comparison of the effects of the sampling methods on the datasets with a minor imbalance problem.

	Pre Process	**None**	**Under**	**Over**	**SMOTE**
Partner 3	False	43.1%	43.3%	43.4%	44.5%
	True	46.6%	46.4%	46.6%	47.6%
Partner 1	False	59.7%	59.8%	59.7%	59.4%
	True	62%	63.4%	63.4%	64.4%
Partner 4	False	60.8%	64.7%	61.9%	63.6%
	True	58.3%	63%	61.0%	60.6%

Table 4.4: Overview of data-sampling methods to combat imbalance for dataset with a minor imbalance problem.

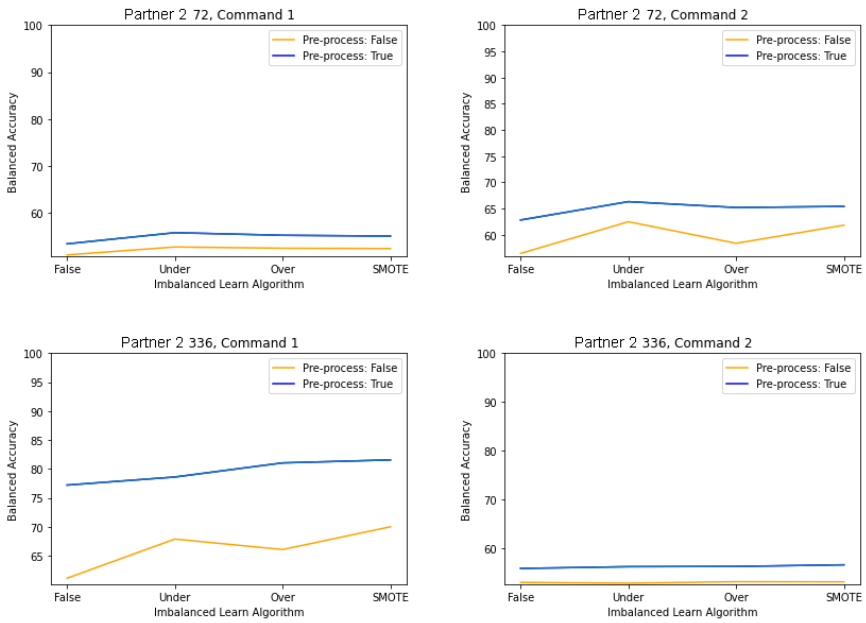

Figure 4.5: Comparison of the effects of the sampling methods on the dataset with various time horizons.

Lastly, the analysis of the Partner 2 dataset is limited only to the best performing variations of the dataset - namely, 72 and 336 hours one. The comparison is given in Figure 4.5.

First off, the 72-hour time horizon dataset performs better for command 2, but still underperforms for command 1, compared to the 336-hour one. The choice of the sampling method is different as well - Partner 2 72 performs better when using Under sampling, while for Partner 2 336 SMOTE has a bigger positive impact.

The overview is given in Table 4.5.

	Preprocess	**None**	**Under**	**Over**	**SMOTE**
Partner 2 72 Command 1	False	51.1%	52.8%	52.5%	52.4%
	True	53.4%	55.8%	55.2%	55.1%
Partner 2 72 Command 2	False	56.4%	62.4%	58.3%	61.8%
	True	62.8%	66.3%	65.1%	65.4%
Partner 2 336 Command 1	False	61.1%	67.8%	66%	70%
	True	77.2%	78.5%	81%	81.5%
Partner 2 336 Command 2	False	53%	52.8%	53.1%	53.1%
	True	55.8%	56.2%	56.3%	56.4%

Table 4.5: Overview of the effects of the sampling methods on the dataset with various time horizons.

As the results of this subchapter show, for most of the datasets the combination of feature aggregation coupled with data sampling to combat imbalance in the dataset work well to increase the performance of the base classifiers. Some datasets this effect is strong - for example, Partner 2 336 shows 20% increase in accuracy, while for others this number is a lot smaller.

Partner 4 dataset is the only dataset that deviates from the behavior established by other datasets. Turning the preprocessing off while still using data sampling methods increases the accuracy, but only by so much. Again, the high dimensionality coupled with low example number could be the reason behind this behavior.

4.0.4 TPOT

Having found the best parameters for feature engineering and data-sampling methods, TPOT was the next step. AutoML algorithms are intended to run for long periods of time - hours to days to find good, if not the best machine learning pipeline for the problem at hand [29].

The overview of the parameters is given below in Table 4.6:

The discussion about generations and population size, as well scoring approach was given earlier in the TPOT subchapter. There are two new parameters introduced here:

1. Cross Validation Fold - the number of cross validation folds done in the optimization process;

2. Early stop - TPOT will stop the search if there are no improvements over this number of generations;

TPOT run parameters	
Generations	100
Population size	100
Cross Validation Fold	8
Scoring	Balanced accuracy
Early Stop	5

Table 4.6: Overview of the TPOT parameters.

	No preprocessing	Best score without TPOT	TPOT pipeline testing
Partner 1	58.5%	64.4%	68%
Partner 2 72	Command1: 51.1%	55.8%	57.8%
	Command2: 62.8%	66.3%	67.4%
Partner 2 336	Command1: 61.1%	81.5%	86.7%
	Command2: 53%	56.4%	57%
Partner 3	43.7%	47.6%	48%
Partner 4	60.3%	64.7%	67.8%
Partner 5	37.6%	39.9%	56%
Partner 6	43.9%	50.2%	55%

Table 4.7: Overview of the TPOT pipeline results.

The rest of the parameters were left as default and can be found at [29]. In this project, depending on the size of the dataset under testing, the run time of the search could take anywhere between 8 to 48 hours, depending when the early stop got triggered.

Table 4.8 presents the evaluations found by TPOT, as well as the base scores and best scores attained so far, using the combination of preprocessing and data-sampling methods:

TPOT has indeed found better models across the board for all datasets, although some results are less exciting than the others. These pipelines, along with the optimized parameters can be found in Appendix. These pipelines include the preprocessing techniques for the data and the machine learning model(s) used in the training. However, the code omits the feature engineering and data sampling methods steps that were found to be the most useful. Hence, to get the full potential out of these pipelines, these methods should be used beforehand.

An overview of found pipelines is given below, per partner and command in the dataset. Details about the particular preprocessing methods and models are beyond the scope of this thesis, and can be viewed on *scikit-learn* [26].

Partner 1

- Preprocessing algorithms:
  - Standard Scaler;
  - Normalizer;
- Stacking estimator - Gradient Boosting Classifier;
- Model - Decision Tree Classifier;

Partner 2 72-hour horizon

Command 1

- Preprocessing algorithms:
  - Variance Threshold;
- Stacking estimator - Gaussian Naive Bayes;
- Model - K-Nearest neighbors;

Command 2

- Preprocessing algorithms:
  - Normalize;
- Stacking estimator - Multi-Layer Perceptron;
- Model - Extra Trees Classifier;

Partner 2 336-hour horizon

Command 1

- Preprocessing algorithms:
  - Polynomial Features;
  - One Hot Encoder;
  - Principal Component Analysis;
- Stacking estimator - Stochastic gradient descent;
- Model - Gradient Boosting Classifier;

Command 2

- Preprocessing algorithms:

  - Polynomial Features;
  - Principal Component Analysis;
- Stacking estimator - Bernoulli Naive Bayes;
- Model - Random Forest Classifier;

Partner 3

- Preprocessing algorithms:
  - Polynomial Features;
  - Principal Component Analysis;
  - Robust Scaler;
  - Normalizer;
  - Max Abs Scaler;
  - Function Transformer;
- Model - Extra Trees Classifier;

Partner 4

- Preprocessing algorithms:
  - Recursive Feature Elimination;
  - Robust Scaler;
  - Normalizer;
- Stacking estimator - Decision Tree Classifier and Extra Trees Classifier;
- Model - K-Nearest neighbors;

Partner 5

- Stacking estimator - Random Forest Classifier and XGB Classifier;
- Model - K-Nearest neighbors;

Partner 6

- Preprocessing algorithms:
  - Recursive Feature Elimination;
  - Standard Scaler;
  - FastICA;
- Model - Extra Trees Classifier;

	Objective value	Calculation time	Non-physical spill
Partner 1	-0.004%	-2%	100%
Partner 2 72	-0.0017%	-4%	95%
Partner 2 336	0.02%	-202%	91%
Partner 3	0.0004%	-2%	25%
Partner 4	-0.27%	-413%	97%
Partner 5	0.02%	4%	NA
Partner 6	-0.002%	56%	NA

Table 4.8: Summary of the performance evaluation of machine learning pipelines.

4.0.5 Comparison between default settings and ML-predicted commands.

To evaluate these pipelines properly, their results, e.g. predicted commands, are passed onto Short-term Hydro Optimization Program (SHOP) by SINTEF, a modeling tool for hydro operation planning [3]. These simulations will indicate how these commands affect the work of the hydropower plants. At the same time, the default setting for these watercourses are also simulated. The difference between the two are then compared.

Depending on the type of the commands in the dataset, three objectives can be tested:

1. Objective value - the result of the optimization model, indicating capital saved.
2. Calculation time - time SHOP used to get the optimal result for each case.
3. Non-physical spill - the percentage indicates how many cases of non-physical spill have been avoided. Non-physical spill occurs when the water starts to spill before the reservoir is filled up.

Positive values are in favor of the results obtained by machine learning pipelines. For the purpose of avoiding non-physical spill, the results predicted by machine

learning pipelines would have prevented between 25% to 100% of these cases, compared to the default settings. Also, as it was pointed out when discussing the performance on datasets with different time horizons, the higher time horizons perform a lot better at predicting the non-physical spill.

When it comes to the objective value and calculation time, the results vary greatly. That being said, even the tiny increase/decrease in objective values result in thousands if not millions of NOK saved/lost. For confidentiality reasons, the exact value must be omitted.

For more complex watercourses, ML pipeline also offers significant savings in calculation time, but clearly is losing when it comes to simpler ones. This makes ML an excellent choice for complex problems, where CPLEX [3] method struggles, since the faster a proposed command can be acquired, the more time is left for optimization tools to be run.

Chapter 5

Conclusion and Future Work

The results obtained in this project indicate that significant increase in predictive capabilities of the classifiers can be gained if a preliminary processing is done on the dataset and the issue with imbalance is addressed before the data is given to the pipelines. The high feature count coupled with a short number of available historical data, as well as imbalancing of the datasets results in a hard to crack problem. The pipelines found by the TPOT, combined with the proposed preprocessing techniques already show that the predicted command would avoid non-physical spills in the reservoirs, as opposed to the program picking the default setting value.

The interest of the industrial partners in this project was not only to answer the question of whether machine learning can be utilised for the task at hand, but also discovering a process of finding the machine learning pipelines in practice. Utilising the exhaustive search to find best feature extraction and data sampling parameters, coupled with TPOT present a simple methodical approach to finding a good pipeline for hydropower plant scheduling optimisation. This process is automated from tip to toe and is rather error-free since it involves no human interaction. The main drawback is it being computationally demanding and hence time consuming. In practice, this approach does not require to be run often, rather when significant updates to the datasets have been acquired. At the same time, there are a number of commercially available cloud supercomputers that can be used to counter this problem.

5.1 Answering Research Questions

Research questions posed in 1 were designed to guide the work in this thesis. While the findings answer every question, it is beneficial to do so explicitly here as well.

Research Question 1: Are there suitable feature engineering methods that could prepare the datasets for machine learning models?

The structure of the datasets in this project presented a natural way of approaching the feature engineering process by aggregating time-related features together over some interval **n** to decrease the number of features. This approach had a very high impact on the predictive power of the classifiers, despite its simplicity. Empirical evidence in 4.0.1 suggests that this approach indeed is beneficial in all datasets, increasing the evaluation score of the classifiers, more for some than the others. An important find was that the best value of **n** for all but one dataset was the value of 2, indicating the optimal position where information loss was minimal, but nevertheless decreasing the feature count.

Research Question 2: Can the issues with imbalanced datasets be alleviated?

The overview of the datasets in Table 2.2 indicated issues with class representations. To alleviate the potential problems data sampling methods provided by imbalanced-learn library [22] were used. As we showed in 4.0.3 these approaches had positive impacts on prediction accuracy, especially for the datasets that had higher imbalancing problem than the others.

Comparing to similar researches [4] [7] show that in this project the amount of historical data available was quite low. If more data cannot be accumulated, especially for the minority cases, the approach demonstrated in this paper to battle imbalance is a reliable first step.

Research Question 3: Can Automatic Machine Learning improve modeling and inference?

An exhaustive search for best feature engineering settings and imbalanced learn showed considerable improvements compared to using the datasets as is. TPOT, the choice of AutoML algorithm for this project was the next step in looking for best combination of machine learning pipelines and data preprocessing techniques.

In subchapter 4.0.4 we present the best results found by TPOT. These pipelines, without any exceptions were superior to the finds done before. Despite the fact that completeness of the TPOT methods is far from perfect, it was possible to attain decent results with the available tools in its disposal.

5.2 Directions for future work.

Part of the research process is to recognize it flaws and outline the areas for improvement and future work. These are aligned with research questions, given that the main focus was put on answering them.

One of the main shortcomings of the available datasets is the number of the examples. This paired with a high count of features makes it hard for the classifiers to generalize and learn to predict all classes well. Increasing the number of available examples in the datasets would alleviate this problem. In cases where there is not

enough of the available historical data, it could be helpful to investigate ways of generating synthetic data using domain knowledge. As the analysis in Effects of Imbalanced learn subchapters shows, SMOTE data sampling helps to improve the accuracy of the classifier, and it does so by generating new examples using simple interpolation. Paired with domain knowledge could lead to both more examples available, but also solve the issue with the imbalancing of the datasets.

One of the interesting finds was how well data-sampling techniques in the preprocessing has worked for most of the datasets. Taking a more in-depth look at other available techniques, paired with domain expertise could result in better sampling methods that could both help in alleviating the imbalancing issue, as well as boost synthetic samples generation.

On the other note, the scores of the classifiers do not properly reflect the goal of this project, as we pointed out in the previous chapter. Creating a separate scoring function that expresses this more clearly would be useful in properly evaluating the performance of the classifiers. Just as importantly, this new scoring metric would drive the search in TPOT and hyper parameter optimization in a more correct direction - for example fitness function evaluating candidates' proposed commands in avoiding non-physical spill would theoretically find better solutions for these cases.

In this project, although the available datasets are of the same nature, they were treated like separate problems. Having different command types, number of reservoirs, time horizons, etc. have cemented this choice. One of the improvements to the workflow would be to find a way of conjoining the available knowledge, either by finding a way of generalizing the available data to make it analogous for the classifiers, or use Transfer Learning [32] that would utilize the knowledge learned from one of the datasets in training classifier(s) the other dataset.

Lastly, the choice of the AutoML limited the search of the potential solutions, given that it does not include any deep-learning models or any of the newer approaches. One could consider this as an unexplored search space which was untouched by the current research, and as such presents an excellent place to advance the ideas presented in this work.

Bibliography

[1] "Electricity production - energifakta norge." https://energifaktanorge.no/en/norsk-energiforsyning/kraftproduksjon/#hydropower, 2022. Accessed: 2022-05-22.

[2] C. Bordin, H. Skjelbred, J. Kong, and Z. Yang, "Machine learning for hydropower scheduling: State of the art and future research directions.," *Procedia Computer Science*, vol. 176, pp. 1695–1668, 2020.

[3] "Shop - sintef.." https://www.sintef.no/en/software/shop/. Accessed: 2022-05-09.

[4] N. Ehsani, B. Fekete, C. Vörösmarty, and Z. Tessler, "A neural network based general reservoir operation scheme," *Procedia Computer Science*, vol. 30, no. 4, pp. 1151–1166, 2015.

[5] A. Hamann and G. Hug, "Simulation of a hydropower cascade using historical data and state estimation," 2022.

[6] J. e. a. Bernardes, "Hydropower operation optimization using machine learning: A systematic review," vol. 3, no. 1, pp. 78–99, 2022.

[7] R. Naresh and J. Sharma, "Reservoir operation using a fuzzy and neural system for hydrogeneration scheduling," *International Journal of Systems Science*, vol. 32, no. 4, pp. 479–486, 2001.

[8] A. Dariane and A. Moradi, "Comparative analysis of evolving artificial neural network and reinforcement learning in stochastic optimization of multireservoir systems," *Hydrological Sciences Journal*, vol. 61, no. 6, pp. 1141–1156, 2016.

[9] T. Jo, *Machine Learning Foundations.* SPRINGER NATURE, 2022.

[10] P. H. Kaur H. and M. A., "A systematic review on imbalanced data challenges in machine learning," *ACM Computing Surveys*, vol. 52, no. 4, pp. 1–36, 2020.

[11] A. Fernandez, S. Garcia, M. Galar, C. Prati, B. Krawczyk, and F. Herrera, *Learning from Imbalanced Datasets.* SPRINGER, 2019.

[12] O. Okun, G. Valentini, and M. Re, *Ensembles in machine learning applications*. Springer, 2011.

[13] W. Wang, "Some fundamental issues in ensemble methods.," *IEEE World Congress on Computational Intelligence*, 2008.

[14] "Imbalanced-learn.org. 2022. fitting model on imbalanced datasets and how to fight bias — version 0.9.0.." `https://imbalanced-learn.org/stable/auto_examples/applications/plot_impact_imbalanced_classes.html`. Accessed: 2022-05-09.

[15] R. Urbanowicz and J. Moore, "xstracs 2.0: description and evaluation of a scalable learning classifier system," *Evolutionary Intelligence*, vol. 8, no. 2-3, pp. 89–116, 2015.

[16] V. D., W. B., M. A., B. W., R. M., W. S., and M. J., "A balanced accuracy function for epistasis modeling in imbalanced datasets using multifactor dimensionality reduction," *Genetic Epidemiology*, vol. 31, no. 4, pp. 306–315, 2007.

[17] J. Kelleher, B. M. Namee, and A. D'Arcy, *Fundamentals of machine learning for predictive data analytics.*

[18] T. Agrawal, *Hyperparameter optimization in machine learning.* APRESS, 2021.

[19] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[20] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, pp. 1–27, 2021.

[21] S. Garcia, J. Luengo, and F. Herrera, *Data Preprocessing in Data Mining.* SPRINGER, 2015.

[22] "Imbalanced-learn.org. 2022. welcome to imbalanced-learn documentation!." `https://imbalanced-learn.org/`. Accessed: 2022-05-09.

[23] S. Karmaker, M. Hassan, M. Smith, L. Xu, C. Zhai, and K. Veeramachaneni, "Automl to date and beyond: Challenges and opportunities," *ACM Computing Surveys*, vol. 54, no. 8, pp. 1–36, 2022.

[24] M. Wever, A. Tornede, F. Mohr, and E. Hullermeier, "Automl for multi-label classification: Overview and empirical evaluation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 3037–3054, 2021.

[25] D. Simon, *Evolutionary optimization algorithms.* John Wiley & Sons Inc, 2013.

[26] 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*, vol. 12, pp. 2825–2830, 2011.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[28] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 785–794, ACM, 2016.

[29] R. Olson, "Using tpot - tpot.." https://epistasislab.github.io/tpot/using/. Accessed: 2022-05-09.

[30] "Hpc.ntnu.no. 2022. idun – high performance computing group.." https://www.hpc.ntnu.no/idun/. Accessed: 2022-05-09.

[31] S. Russell and P. Norvig, *Artificial intelligence.* Pearson Education, Inc., 3 ed., 2010.

[32] S. Bozinovski, "Reminder of the first paper on transfer learning in neural networks, 1976," *Informatica*, vol. 44, no. 3, 2020.

Appendix A

Partner 1

```
StackingEstimator(estimator=GradientBoostingClassifier(
    learning_rate=1.0, max_depth=4,
    max_features=0.6500000000000001, min_samples_leaf=6,
    min_samples_split=6, n_estimators=100,
    subsample=0.55)),

StandardScaler(),

Normalizer(norm="max"),

DecisionTreeClassifier(criterion="gini", max_depth=2,
                       min_samples_leaf=19,
                       min_samples_split=5)
```

Appendix B

Partner 2 72

Command 1:

```
VarianceThreshold(threshold=0.2),
StackingEstimator(estimator=GaussianNB()),
KNeighborsClassifier(n\_neighbors=92, p=1,
                     weights="uniform")
```

Command 2:

```
Normalizer(norm="l2"),
Normalizer(norm="l2"),
StackingEstimator(  estimator=MLPClassifier(alpha=0.01,
                    learning_rate_init=0.001)),
ExtraTreesClassifier(bootstrap=True, criterion="gini",
                    max_features=0.8500000000000001,
                    min_samples_leaf=10,
                    min_samples_split=20,
                    n_estimators=100)
```

Appendix C

Partner 2 336

Command 1:

```
StackingEstimator(  estimator=SGDClassifier(alpha=0.001,
                    eta0=1.0,
                    fit_intercept=False, l1_ratio=0.75,
                    learning_rate="constant",
                    loss="perceptron",
                    penalty="elasticnet", power_t=0.5)),

PolynomialFeatures( degree=2, include_bias=False,
                    interaction_only=False),

OneHotEncoder(  minimum_fraction=0.1, sparse=False,
                threshold=10),

OneHotEncoder(  minimum_fraction=0.25, sparse=False,
                threshold=10),

PCA(iterated_power=7, svd_solver="randomized"),

GradientBoostingClassifier( learning_rate=0.5, max_depth=9,
                            max_features=0.05,
                            min_samples_leaf=13,
                            min_samples_split=9,
                            n_estimators=100,
                            subsample=0.9500000000000001)
```

Command 2:

```
StackingEstimator(  estimator=BernoulliNB(alpha=0.001,
                    fit_prior=False)),
```

```
PolynomialFeatures( degree=2, include_bias=False,
                    interaction_only=False),

PCA(iterated_power=10, svd_solver="randomized"),

RandomForestClassifier( bootstrap=False, criterion="entropy",
                        max_features=0.2,
                        min_samples_leaf=4,
                        min_samples_split=7,
                        n_estimators=100)
```

Appendix D

Partner 3

```
make_union(
        make_pipeline(
            MaxAbsScaler(),
            PolynomialFeatures( degree=2, include_bias=False,
                                interaction_only=False),
            PCA(iterated_power=3, svd_solver="randomized"),
            RobustScaler(),
            Normalizer(norm="l2")),

FunctionTransformer(copy)),

ExtraTreesClassifier(   bootstrap=False, criterion="entropy",
                        max_features=0.9500000000000001,
                        min_samples_leaf=1,
                        min_samples_split=3,
                        n_estimators=100)
```

Appendix E

Partner 4

```
make_union(
        FunctionTransformer(copy),
        FunctionTransformer(copy)
    ),

Normalizer(norm="l2"),

StackingEstimator(  estimator=DecisionTreeClassifier(
                        criterion="entropy", max_depth=9,
                        min_samples_leaf=20,
                        min_samples_split=18)),

RFE(estimator=  ExtraTreesClassifier( criterion="gini",
                    max_features=0.6000000000000001,
                    n_estimators=100),
                    step=0.35000000000000003),

Normalizer(norm="l2"),
KNeighborsClassifier(n_neighbors=5, p=1, weights="distance")
```

Appendix F

Partner 5

```
StackingEstimator(  estimator=RandomForestClassifier(
                        bootstrap=True,
                        criterion="gini",
                        max_features=0.9,
                        min_samples_leaf=4,
                        min_samples_split=16,
                        n_estimators=100)),

StackingEstimator(  estimator=XGBClassifier(
                        learning_rate=0.5, max_depth=10,
                        min_child_weight=12, n_estimators=100,
                        n_jobs=1, subsample=0.9,
                        verbosity=0)),

KNeighborsClassifier(n_neighbors=1, p=1, weights="distance")
```

Appendix G

Partner 6

```
make_union(
        make_pipeline(
            StandardScaler(),
            FastICA(tol=0.8500000000000001)
        ),

        RFE(estimator=ExtraTreesClassifier(
            criterion="entropy",
            max_features=0.6000000000000001,
            n_estimators=100),
            step=0.8)
),

ExtraTreesClassifier(   bootstrap=False, criterion="gini",
                        max_features=0.3,
                        min_samples_leaf=1,
                        min_samples_split=3,
                        n_estimators=100)
```

Appendix H

Project files

Following repository provides public access to all of project files:

Babayev, Piri "An intelligent decision-making process for hydro scheduling.", 2022. [Online]. Available: https://zenodo.org/record/6590166. [Accessed: 24- May- 2022].