Master's Thesis 2019 30 ECTS
Faculty of Science and Technology

Combining Mathematical Programming and Machine Learning in Electricity Price Forecasting

Eivind Bekken Sveen
Master of Environmental Physics and Renewable Energy

Abstract

The future electricity market will have a higher share of generation from unregulated renewable energy sources because of binding targets for reducing CO_2 emissions. Energy reserves for regulating the power production will be needed, and there will be an increased flow of power across country borders. As a result, electricity prices are expected to become more volatile. Electricity price forecasting models will thus need a higher level of detail and temporal resolution to capture the increased variation in the market.

With increased resolution and complexity, the computational times will increase. This thesis investigates how the complexity of a prototype hydrothermal scheduling model named PriMod under development at SINTEF Energy Research can be reduced, and thus saving computational times, by exploiting patterns in the input data.

The potential for reducing the problem size has been assessed by removing constraints with a dual value of zero in the optimal solution of the linear programming (LP) problems produced by PriMod. Findings show that the number of constraints in the current implementation of the prototype model on two different datasets can be reduced with 77% and 93%, respectively. A genetic algorithm was able to further reduce the number of constraints, while having an insignificant impact on the modelled total costs.

An algorithm for reducing the number of constraints by combining water balances is presented. The algorithm was used with an LP problem from PriMod and proved effective in reducing the number of water balances while having an insignificant impact on the total system cost. After combining the water balances, the number of water balances were reduced by 64%.

Finally, a framework for reducing the problem sizes is presented. In this framework, constraints that can be removed are identified and used as target data to train a machine learning model. The framework was tested using an artificial neural network to predict which constraints can be removed, based on data on inflow, wind power production and load. The mean computation time of 1092 LP problems generated by PriMod was reduced by 55%, while the mean percentage error in total system cost was 0.14%. The machine learning task is a multi-label classification problem, which is complex and requires more training data and models with more parameters than the one presented in thesis, to achieve higher prediction accuracies.

Sammendrag

Fremtidens elektrisitetsmarked vil ha en større andel uregulerbar fornybar energi som følge av krav om lavere CO_2-utslipp. Det vil være økt behov for energireserver, og det vil være økt lastflyt på tvers av landegrense. Elektrisitetspriser er forventet å være mer flyktige. Modeller brukt for å gi prisprognoser på elektrisitet vil trenge høyere detaljnivå og tidsoppløsning for fange opp de økende variasjonene i markedet.

Beregningstider øker med oppløsningen og kompleksiteten til modellene. I denne avhandlingen undersøkes hvordan beregningstidene til den hydrotermiske driftsmodellen PriMod under utvikling ved SINTEF Energi kan reduseres ved å utnytte mønstre i inngangsdataene.

Det er vurdert hvor mye problemer basert på lineærprogrammering (LP) fra PriMod kan reduseres ved å fjerne restriksjoner som har en dualverdi lik null i de optimale løsningene. I den nåværende implementeringen av modellen kan antall restriksjoner reduseres med henholdsvis 77% og 93 % benyttet på to forskjellige datasett. En genetisk algoritme kunne redusere problemstørrelsen ytterligere uten at de beregnede systemkostnadene ble påvirket i betydelig grad.

En algoritme for å redusere antall restriksjoner gjennom å slå sammen vannbalanser er vist frem. Algoritmen ble benyttet på et LP-problem fra PriMod, og kunne redusere antall vannbalanser i problemet med 64%.

Et rammeverk for å redusere størrelsene til problemene er presentert. I dette rammeverket identifiseres restriksjoner som kan fjernes. Disse brukes som måldata til å trene en maskinlæringsmodell til å kunne forutsi hvilke restriksjoner som kan fjernes. Rammeverket ble benyttet med et nevralt nettverk som ga prediksjoner basert på data for tilsig, vindkraftproduksjon og last. Den gjennomsnittlige beregningstiden til 1092 LP-problemer fra PriMod ble redusert med 55% med en feil i total systemkostnad på 0.14%. Maskinlæringsproblemet er et klassifiseringsproblem hvor hver instans kan knyttes til flere klasser. Dette er et komplekst problem og det er behov for mer treningsdata og modeller med flere parametere enn det som vises i denne avhandlingen for å kunne gi mer nøyaktige prediksjoner.

Contents

Chapter 1

Introduction

1.1 Background

The future European electricity network will have a larger share of unregulated renewable power generation than today [17] because of strict goals of reduction in CO_2 emissions set by the European Union. By 2030, the European commission has a goal of reaching a 40% cut in CO_2 emissions compared to 1990 levels [19]. In 2016, the share of total power production from wind and solar in Europe was 17% [14]. This was more than a doubling of the installed capacity in 2010. In the reference scenario presented by the European Union commission, this share will increase to 25% in 2030, and 39% in 2050 [15]. At the same time, there is an ongoing increase in the capacity of the electricity grid [17], and electricity is expected to play a larger role in the future European energy mix, especially in the transport sector and for heating.

As a result, there will be a growing demand for system services to ensure network frequency and voltage stability. These services are expected to become a large part of the future European energy market [16] and significant energy reserves for balancing the electricity production will be needed. Hydropower is flexible and is already used for balancing production from wind turbines. Hydropower in Norway is used for balancing wind production in Denmark, and hydropower in Canada provides balancing for regions with a high share of wind production in the US [27].

Because of the changes in the generation mix, there will be an increased flow of power across nation borders [16], in particular between Great Britain, Norway, Denmark and Germany. Cables between Norway and Great Britain, as well as between Norway and Northern Germany are currently under construction [1]. The European Council have set interconnection targets requiring that the transmission capacities from one member state to another allows 10% of the electricity produced in the country to be transported across the border [18].

The hydrothermal systems are complex, and the modelling of such systems is a non-trivial task. The increased penetration of renewable energy sources, and the tighter integration of the electricity network, places a challenge in modelling the electricity power market and in creating reliable electricity price forecasts. Forecasting of electricity prices and modelling of the power market is important for system operators in the operation, planning and development of the electricity distribution network. It is important for power producers as well, for planning power generation and evaluating future investments and expansions. Electricity prices in the physical markets are expected to become more volatile [50], which will have a significant impact on how the hydropower plants and reservoirs will be operated [33]. This creates a need for electricity price forecasting models with finer temporal resolution that adequately captures the increased variation in the market.

Fundamental market models based on mathematical and stochastic programming are widely used for hydrothermal generation scheduling and electricity price forecasting in the Nordic electricity market [21, 23, 32]. Models as in e.g. [30], forecast prices through simulating the hydrothermal energy system. In addition to give a price forecast on electricity, the models can find marginal prices on stored hydro reserves.

By comparing two hydrothermal scheduling models with different levels of detail and temporal resolution, the authors in [28] shows that it will be increasingly important to model short term aspects due to the increased penetration of unregulated renewable energy sources and the tighter integration of the European electricity network. In addition, the same authors point out the importance of having a detailed representation of the hydrothermal system.

In [11], the effect of increasing the time resolution in modelling power system with significant levels of renewable generation is studied. It is shown that capturing more of the variation in power generation and system load leads to higher and more realistic estimated costs. At the same time, more detailed models can more accurately model the benefits that comes with available and flexible hydropower capacity [28].

In [29], the authors describe the need for a higher level of detail in thermal modelling in hydrothermal price forecasting models. The hydropower production increases slightly due to the need for regularizing thermal power. This coincides with the results in [11], that show how increased temporal resolution is needed to capture the inflexibilities of thermal units.

With the need for increased complexity and resolution, the challenge of computational performance becomes more prominent. Machine learning methods can be used to create generalized models that overcome the computational performance issues, but at the cost of the precision found in models based on mathematical programming. A myriad of different approaches exist for predicting electricity prices with probabilistic methods [64], including deep neural network [37, 56]. Most approaches focus on forecasting day-ahead electricity prices with an hourly resolution on a short time scale.

Machine learning methods such as artificial neural networks has also been used to solve general mathematical programming problems [39]. A motivation for using neural networks in such applications is the approximation theorem of neural networks, which states that a neural network with at least one hidden layer and a finite number of activation units (neurons) can be created to approximate the solution of mathematical functions, assuming a continuous function of finite dimensional space [24].

A drawback of using machine learning or generalized models in electricity price forecasting, is that it does not yield detailed information about the simulated system. Examples of useful information is marginal prices on all energy products and information on limiting elements, i.e. environmental restrictions or cable connections that acts as bottlenecks. On the other hand, machine learning methods has the promise of creating effective generalized models that can assist models based on mathematical programming in finding optimal solutions in optimization problems. By instead using machine learning to assist mathematical programming models, the information on the system may still be retrieved, while the machine learning model provides increased computational efficiency. The theme of using machine learning in combination with mathematical programming has been explored in [3]. The article states that machine learning and mathematical programming has become increasingly intertwined, as the computational efficiency of traditional solution algorithms for solving mathematical programming problems become inadequate when the model complexity grows large.

1.2 Scope

In this thesis, the challenge of computational performance in hydrothermal scheduling models with an increasing level of detail, is addressed. New ground is threaded to find if machine learning can be used to assist models based on mathematical programming. The goal is to find whether the size and complexity of an electricity price forecasting model under development can be reduced. Machine learning may be used to recognize patterns in the input data, and can potentially suggest mathematical constraints that can be removed from the optimization problems without impacting the result of the optimal solution.

The work first consists of identifying the parts of the problem that can be reduced. It is then investigated whether a machine learning model can be trained to suggest how the problem can be reduced, before the problem is solved with existing mathematical programming solution methods. An exploration into multi-label multiclass classification is made, and a general framework for using machine learning in reducing generation scheduling problems based on linear programming is presented.

The present work is conducted with a prototype short term hydrothermal scheduling model named PriMod, which is part of the ongoing PRIBAS project [31]. Focus is on the specific input parameters inflow, wind power generation and load.

1.3 Outline

Theory

Chapter 2 will begin by giving a background on linear programming used in hydrothermal scheduling, including the short term operational model PriMod. Especially relevant is the theory behind dual values, that will be used to find ways to reduce the linear programs. Next, there is a section on the theory behind genetic algorithms, that is used to further reduce size of the linear problems.

An overview on machine learning, with focus on multi-label multiclass classification problems, is given in section 2.3. Some insight into different approaches are given, as well as some recent research in this area. The main focus, however, will be artificial neural networks and the use of this machine learning method in multi-label classification.

Finally, an overview on electricity price forecasting, hydrothermal generation scheduling and the ongoing PRIBAS project at SINTEF Energy Research is given in 2.4. A prototype operational model of the PRIBAS project, called PriMod, is described. The work is also placed in a larger context, presenting some other current approaches in electricity price forecasting.

Method

Chapter 3 will start with giving a description of the case used with the PriMod scheduling model. The next section outlines what software is used for solving the linear programs, and the different metrics that are used for evaluating the computational performance.

An approach for reducing the problem size by finding water balances that can be combined is given in section 3.4. Following this, the genetic algorithm used for reducing the problem size is described. After that, an artificial neural network for classifying constraints that can be removed, based on the dual values of the optimal solution and the results from the genetic algorithm, is shown. A framework for using machine learning to reduce the computational times of PriMod is presented in the last section of this chapter.

Results

In chapter 4, the results from applying the approaches described in chapter 3 on a set of linear programming problems are presented. The LP problems are mainly from the short term operational model PriMod. Results and analysis on the following items are presented in this chapter:

- removal of constraints with dual values of zero

- impact of inflow, wind power production and load on water balances and cable ramping restrictions
- problem reduction by merging water balances
- problem reduction with genetic algorithms
- identification of removable constraints with artificial neural networks

Discussion

In chapter 5, the potential for reducing computational times is discussed, as well as possible ways to improve the machine learning model.

Conclusion

Conclusions on the analysis from the two previous chapters are given in chapter 6. Possible improvements to the machine learning models and future work is outlined in the final section.

Chapter 2

Theory

2.1 Linear Programming

Linear programming consists of optimizing a linear objective function, often called a cost function, under a set of constraining conditions that are expressed as a system of linear equations [42]. The standard form of a linear program is

$$\text{minimize} \quad \mathbf{c}^T\mathbf{x} \tag{2.1a}$$

$$\text{subject to} \quad \mathbf{Ax} = \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0} \tag{2.1b}$$

where $\mathbf{c}^T\mathbf{x}$ is the cost function, $\mathbf{c} \in \mathbb{R}^m$ is a coefficients vector and $\mathbf{x} \in \mathbb{R}^m$ is the corresponding vector of variables [42]. The constraints are defined in (2.1b), where $\mathbf{A}$ is an $m \times n$ coefficient matrix and $\mathbf{b} \in \mathbb{R}^n$ is a vector of scalars.

2.1.1 Feasible Solutions

A *feasible solution* of a linear programming problem is a solution that fulfils the constraints in (2.1b). A *basic solution* to (2.1) is defined as a solution where $\mathbf{A}$ is arranged such that the m first columns are linearly independent, and the $n - m$ last elements of $\mathbf{x}$ are set equal to zero [42]. Since the first m columns of $\mathbf{A}$ are linearly independent, these columns constitute an $m \times m$ nonsingular matrix. This matrix is called a basis of the linear program [42].

The fundamental theorem of linear programming ensures that only basic feasible solutions, i.e. solutions that are both basic and feasible, need to be explored to find an optimal feasible solution of the problem [42]. Furthermore, if $\mathbf{A}$ in (2.1b) has rank m, the solution set is a convex polytope [42]. It can then be shown that a vector $\mathbf{x}$ is an extreme point of this set if and only if this vector is a basic feasible solution [42]. From the two previous results, it follows that there is a finite number of feasible solutions to investigate when solving a problem.

By performing elementary row operations on the system of linear equations, the basis of the problem can be changed to yield different basic feasible solutions. However, a new linear system that ensues from changing the basis is still equivalent to the original problem.

2.1.2 Reduced Cost Coefficients

Having defined the cost function of a linear program as in (2.1), let $z_j = \mathbf{c}^T \mathbf{a}_j$, where $\mathbf{a}_j$ is the jth column of the matrix $\mathbf{A}$. The reduced cost coefficients are then defined as $r_j = c_j - z_j$, where x_j and c_j are the jth elements of $\mathbf{x}$ and $\mathbf{c}$, respectively [42]. The reduced cost coefficients can be used to determine which variables should be in the basis. If a basis for (2.1) has been constructed, then the reduced cost coefficient r_j indicates whether it is beneficial to let a nonbasic variable x_j enter the basis or not. A negative reduced cost coefficient means that increasing x_j from zero to a positive value would reduce the cost function. Thus, introducing x_j into the basis will lead to a more optimal value. All basic variables have a reduced cost of zero.

2.1.3 Dual Values

For all linear programs formulated as (2.1), there is an associated problem formulation where the goal is to maximize a cost function [42]. The associated problem is called the dual of the original problem, and can be stated as follows:

$$\begin{aligned} \text{maximize} \quad & \mathbf{y}^T \mathbf{b} && (2.2a) \\ \text{subject to} \quad & \mathbf{y}^T \mathbf{A} \leq \mathbf{c}^T, \quad \mathbf{x} \geq \mathbf{0} && (2.2b) \end{aligned}$$

where $\mathbf{y}$ are variables of the new problem. The original problem, expressed by (2.1), is then referred to as the primal. An important result in linear programming theory, called the duality theorem of linear programming [42], is that the optimal objective value for the primal is also the optimal objective value of the dual. Thus, if $\mathbf{x}$ is an optimal solution of the primal, then there is a $\mathbf{b}$ that optimizes the dual, and the relationship between these two solutions is

$$z = \mathbf{y}^T \mathbf{b} = \mathbf{c}^T \mathbf{x} \tag{2.3}$$

The primal and the dual will have the same optimal basis. If $\mathbf{b}$ is perturbed slightly, then the optimal basis will not change. From this it follows, that for a small change $\Delta\mathbf{b}$, the change in the cost functions is

$$\Delta z = \mathbf{y}^T \Delta\mathbf{b} \tag{2.4}$$

This equation expresses the change in the objective value relative to the change of the right-hand side in equation (2.1). The elements of $\mathbf{y}^T$ are called the dual values [42] and can be interpreted as the gradient of the cost function. When the constraints are representing monetary costs, the dual values represent marginal prices. In other words, when an optimal solution is found, a dual value indicates the impact a change in the right-hand side of the corresponding constraint will have on the objective value. If a constraint has a dual value equal to zero, altering the right-hand side of this constraint will have no impact on the objective value in the optimal solution. In this case, the constraint can in fact be removed entirely without changing the solution of the LP problem.

2.1.4 Linar Programming Solution Methods

Several methods have been developed for solving linear programs [42]. Two methods are presented in this section.

The *simplex algorithm* determines which basic feasible solution, or extreme point, to evaluate next in order to reduce the objective value. To find which variable to consider next, the algorithm calculates the reduced cost coefficients r_j. If the reduced cost of a variable is negative, this variable will enter the basis. The algorithm then continues by performing operations called pivoting. Pivoting is closely related to Gaussian elimination and consists of changing the system of linear equations to let the chosen variable become basic. At the same time, an existing basic variable will become nonbasic. However, if all the reduced cost coefficients are positive, then the optimal solution has been found, and the algorithm terminates.

Another method for solving linear programs is the *dual simplex method.* The standard simplex method can be used to solve both the primal and the dual, but the dual simplex method optimizes the dual cost function while working with the constraints of the primal. Instead of using the reduced cost coefficients to find which row to pivot such that the cost function of the primal is reduced, the dual simplex method finds which column of $\mathbf{A}$ should enter the basis such that the cost function of the dual increases.

A basic solution is called degenerate if one or more of the basic variables are zero [42]. With highly degenerate problems, there is an increased risk that the simplex method stalls [36]. To overcome this problem, a solving algorithm may add perturbations to the variable bounds. The solver then tries to solve the perturbed problem. If the perturbed problem is successfully solved, the solver removes the perturbations to yield an answer with respect to the original data.

2.1.5 Mixed Integer Linear Programming

If some of the variables in a linear programming problem are required to be an integer, the problem is converted to a mixed integer linear program. A motive for requiring some of the variables to be integers is found in scheduling problems, where certain conditions and decisions may be represented with binary variables.

2.2 Genetic Algorithms

A genetic algorithm is a heuristic method that is useful when there are several objectives that needs to be optimized [2]. Genetic algorithms are inspired by the Darwinian theory of evolution, and much of the underlying principles are the same. In the theory of evolution, strong specimen that are more fit to the surrounding will have a greater chance of reproducing, and the genes of these individuals will therefore have a greater chance of being passed on to future generations. After a sufficient amount of time, certain combinations of genes will start to dominate the population of the species. During the evolution, there is also a chance that mutations will occur, which means that random changes occur in a set of genes. If the changes are advantageous, these genes are likely to be carried over to the next generation.

In a genetic algorithm, a solution vector corresponds to a set of genes, i.e. a chromosome. A solution vector is called an individual and the entire set of solution vectors is the population. A simple genetic algorithm is seeded with a start population, which can be randomly generated. The initial population is then evaluated before the algorithm enters the first generation of the iterative process. Individuals that lead to a more optimized objective function is deemed to be more fit than others. For each generation in the loop, new individuals are created and added to the population through the processes of crossover and mutation. The fitness of the individuals in the expanded population are then evaluated, and a selection procedure is used to carry the most fit individuals over to the next generation.

A variation of the simple algorithm presented above, is the (μ, λ) algorithm [2]. In this algorithm, λ offspring are produced from μ individuals, with the number of offspring larger than the number of parents. The (μ, λ) algorithm differs from the previous in that the next population is only generated from the offspring.

In the crossover operation, two fit individuals are combined to create a new individual. The goal is to create offspring with even better features than the individuals of the current generation, based on a combination of the features of the most fit individuals, such that the algorithm converges towards an optimal solution. The best known forms of crossover operations in genetic algorithms are one-point, two-point and uniform crossover. In one-point crossover, a common index of the two parent vectors is chosen at random, and all values from this index are swapped between the two vectors. In two-point crossover, two indices are chosen at random and the values before and after the two points, respectively, are swapped. Lastly, in uniform crossover, individual values at random indices are swapped.

Mutations create small random changes in the some of the solutions. This is implemented such that each value in a vector has a small chance of being changed. Mutations usually has less impact than the crossover operations, but are helpful in avoiding that the algorithm get stuck in a local optimum [2].

In multi-objective problems, the objectives may be conflicting, and optimizing one objective could mean that another is degraded. Multi-objective problems with conflicting objectives will thus yield solutions that optimizes the different objectives to various degrees. Thus, searching for an optimal solution involves investigating a set of solutions that are said to not be dominated by any of the other solutions. A solution is not dominated if, for each objective, no other solution manages to optimize the objective even further without degrading at least one of the other objectives. Solutions that are not dominated by any other solution are said to be Pareto optimal. The set of Pareto optimal solutions is called the Pareto front.

A concept called elitism is employed in some genetic algorithms. In genetic algorithms with a single objective, elitism means that the most fit individual of a generation is guaranteed to be carried over to the next generation. In the context of multi-objective genetic algorithms, all non-dominated solutions are elite [38].

Much work has been done on multi-objective genetic algorithms and a summary of several well known multi-objective genetic algorithms is given in [38]. A selection algorithm that is well tested and has proven to be effective is the elitist Non-dominated Sorting Genetic Algorithm (NSGA-II) [12]. Rather than focusing on finding samples that optimizes the cost function, NSGA-II focuses on preserving non-dominated individuals close to the Pareto-optimal front. In this approach, each individual in the population is ranked according to its dominance, and several non-dominated fronts are created. It is an elitist approach because all non-dominated solutions are guaranteed to be carried over to the next generation. NSGA-II also use a crowding distance method that aims to find a Pareto front with evenly spread solutions [12].

2.3 Machine Learning

The goal in linear programming is to find an exact solution to an optimization problem. Finding an exact solution can come at the expense of computational performance and model complexity. Machine learning, on the other hand, seeks to model complex problems with models that generalizes well and have high computational performance, but at the cost of the analytical accuracy found in linear programming [3]. Machine learning also solves optimization problems, where the function to be minimized measures the model error. In both linear programming and machine learning, the function to be optimized is often called the objective function or the cost function. In machine learning, yet another name for this function is the loss function.

Machine learning models can be placed into two categories: supervised and unsupervised learning. In supervised learning, samples with known targets are used to train models to predict the target in samples where the target value is unknown. Predictions are based on associated input data called features. Each feature is connected to one specific observable measure, i.e. one particular inflow value, the total wind production in an area, or the daily load profile in an area. In unsupervised learning, machine learning is used to categorize data or find patterns in the input data, without the use of known targets to train the model.

2.3.1 Multi-Label Classification

Classification is a subset of machine learning where features of a sample is used to determine a categorical class that this sample belongs to. When the classification problem consists of several classes, the classification problem is termed a multiclass classification problem. These problems are again divided into single-label and multi-label classification. Multi-label classification differs from the single-label variant in that each sample can be assigned several labels.

Multi-label classification is a field of study that has received much attention in the last decade [61]. A literature study into what methods exist had to be made, and small overview of some of the different approaches is given here.

The simplest approach of multi-label classification is the one-against-all technique, also termed binary relevance [55]. In binary relevance, one machine learning model is created for each label, and the models are trained separately. One downside with this approach is that dependencies between labels are not taken into account. Furthermore, the approach is impractical because in several applications, thousands to millions of classifiers would need to be trained [4, 59], which is slow and requires a large amount of computer memory.

The problem can be transformed by dividing the label space with clustering methods, co-occurrence graphs or random divisions [53]. Predefined divisions can also be used when there is prior knowledge about the data. Results can then be predicted by combining the results of each classifier, or by using an ensemble approach, where a label is assigned to a sample if the label is predicted in the majority of the classifiers.

In label space embedding techniques, the labels are mapped to a smaller embedded space. In the embedded space, the labels are represented as vectors of continuous variables instead of discrete classes. A model must then be trained to predict the label assignment based on the embeddings. Two state-of-the-art approaches are the Cost-Sensitive Label Embeddings (CLEMS) [35] and Label Network Embeddings Multi-Label Classification (LNEMLC) [53].

Multi-label problems can also be tackled by adapting existing machine learning methods for use in multi-label settings. Two such examples are the MLkNN, that is based on the k-Nearest Neighbors method [60], and MLTSVM, a multi-label twin support vector machine presented in [8].

Deep neural networks with Keras [9] and Tensorflow [26] can be used for multi-label classification by choosing an appropriate loss function, and appropriate activation units in the output layer. The authors of scikit-multilearn, a Python library providing solutions for multi-label learning, point out that deep neural networks often lack robustness and need to be specifically tailored for each concrete problem [53]. On the other hand, neural networks can be trained in batches, avoiding some of the memory limitations connected with other methods. Furthermore, neural network implementations allow computers to effectively utilize graphical processing units when training the model.

multi-label classification methods have been used in disease diagnosis, bioinformatics, text and image classification [25]. multi-label classification has also been used in protein function classification [51], where near perfect accuracies are achieved with close to 1000 classes using deep neural networks.

When the number of labels grows large, the problem is often termed extreme multi-label learning (XML). There is no precise definition of how many labels constitute an extreme model, but XML problems often have thousands of labels, and XML models have recently started to tackle problems that consist up to millions of labels [62]. Most extreme multi-label classification methods falls into one of two categories: embedding based methods and tree based methods [41]. Recent work falls into a third category, namely deep learning methods. In [41], a deep learning neural network architecture (XML-CNN) is proposed for use in extreme multi-label text classification. In [62], a framework using deep learning together with non-linear feature space reduction is presented. Both deep learning approaches compared favorably with other XML approaches. In XML, computational performance and scalability becomes important. However, XML is often only concerned with finding the most relevant labels, e.g. for finding the most relevant tags for a text or an image, and less concerned with classifying all labels correctly.

2.3.2 Evaluation Metrics

Measuring accuracy in multi-label classification problems is not as straight-forward as in single-label multiclass or single-class problems, and accuracy measures for single-label classification cannot be applied directly to multi-label classification. There are several forms of multi-label accuracy metrics [25], and a selection of these, used when evaluating the machine learning models in section 3.6, will be described here.

In binary classification, accuracy can be measured by using the *exact match ratio*, that is found by calculating the ratio of correctly predicted samples to the total number of samples [63]. In multi-label classification, the exact match ratio is a harsh form of measurement, because it only measures whether all the labels of each sample were correctly classified or not. If a prediction is mostly correct, e.g. 95 % of the labels were correctly predicted, the exact match ratio will nonetheless yield an accuracy score of zero.

According to [25], there are two groups of classification metrics for multi-label problems: label-based and instance-based metrics.

Label-Based Metrics

Label-based metrics are based on averaging any form of binary classification metric across all the different labels. There are two ways of averaging: micro and macro averaging. In micro averaging, the number of true positives (tp), false positives (fp), true negatives (tn) and false negatives (fn) are all averaged before evaluating the binary metric.

$$B_{\text{micro}} = B\left(\sum_{i=1}^{n} \text{tp}_i, \sum_{i=1}^{n} \text{fp}_i, \sum_{i=1}^{n} \text{tn}_i, \sum_{i=1}^{n} \text{fn}_i\right) \tag{2.5}$$

where B can be any type of binary evaluation metric, and n is the number of labels. In macro averaging on the other hand, the binary score is evaluated for each label, before calculating the mean of all the binary scores.

$$B_{\text{macro}} = \frac{1}{n}\sum_{i=1}^{n} B\left(\text{tp}_i, \text{fp}_i, \text{tn}_i, \text{fn}_i\right) \tag{2.6}$$

Instance-Based Metrics

Instead of averaging across all the labels, instance-based metrics evaluate each sample in the validation set and averages across all the samples in this set. One such metric is the Hamming loss [25], given by

$$\text{Hamming loss} = \frac{1}{m}\sum_{i=1}^{m} \frac{1}{n}|\hat{Y}_i \triangle Y_i| \tag{2.7}$$

where m is the number of samples, $\hat{Y}_i$ is the set of predicted labels for sample i, Y_i is the set of true labels, $1/n$ is a normalizing factor with n being the total number of labels, and $\triangle$ denotes the symmetric difference.

An instance-based accuracy metric is the multi-label accuracy (ACC), that is defined as the average number of correctly classified positive labels divided by the total number of positive labels [25]:

$$\text{ACC} = \frac{1}{m}\sum_{i=1}^{m} \frac{|\hat{Y}_i \cap Y_i|}{|\hat{Y}_i \cup Y_i|} \tag{2.8}$$

Here, m is the number of samples, and $\hat{Y}_i$ and Y_i are the set of predicted and true labels, respectively. If the sets of predicted and true labels are empty, the accuracy is defined to equal unity.

Two other measures are *precision* and *recall*. Precision, also called the *true positive rate*, is defined as the fraction of correctly classified labels to the true labels, whereas recall measures the number of correctly classified labels to the number of predicted labels. The F1-score, defined by equation (2.9), is the harmonic mean of precision and recall [25].

$$\text{F1-score} = \frac{1}{m}\sum_{i=1}^{m} \frac{2|\hat{Y}_i \cap Y_i|}{|\hat{Y}_i| + |Y_i|} \tag{2.9}$$

Receiver Operating Characteristic

To create a receiver operating curve (ROC), the *false positive rates* is needed. The false positive rate is the ratio of false positives to the total number of predicted positives. In an ROC graph, the true positive rates as a function of false negative rates are plotted [46]. The area under the curve (AUC) of the ROC curve is a measure of how well a classification model performs. An AUC score of less than 0.5 means the model performs worse than random guessing, while a prefect score of 1.0 means the model correctly classifies all the labels in all the test samples.

Label Imbalance

A common phenomenon in multi-label classification tasks is that the frequency of the individual labels can be heavily imbalanced [6]. A measurement for assessing the imbalance in multi-label datasets is [6]

$$\text{IRLbl}(Y) = \frac{\max_{i=1}^{m}\left(\sum_{j=1}^{n} Y_{ij}\right)}{\sum_{j=1}^{n} Y_{ij}} \tag{2.10}$$

where Y denotes a binary label set with m samples and n labels. A mean *IRLbl* value higher than 1.5 should be considered as imbalanced [6].

2.3.3 Artificial Neural Networks

The basis of a neural network is the artificial neuron [46]. A neuron consists of a non-linear activation function that is given a weight, and the neural network can be trained by finding weights such that a chosen loss function is minimized. Several layers of neurons can be connected, which results in what is called a deep network. All layers except the input and output layers are called hidden layers. When the layers are densely, or fully, connected, each neuron in one layer is connected to every neuron in the other layer, and each connection is assigned a weight that is updated during training.

When choosing a loss function for the artificial neural network, the main criteria is that the function can be minimized, and that it has a continuous derivative. In multi-label classification, binary cross-entropy has proved to be an effective loss function [43, 41]. The loss-function based on binary cross-entropy is defined as follows [46]:

$$\min_{\Theta} -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}\left[y_{ij}\log\left(\phi(z_{ij})\right) + (1-y_{ij})\log\left(1-\phi(z_{ij})\right)\right] \tag{2.11}$$

where Θ denotes the model hyperparameters, y_{ij} is a binary value indicating whether sample i belongs to class j and $\phi(z)$ is the activation function. z_{ij} is defined as $\mathbf{x}_i^T \mathbf{w}_{ij}$, where $\mathbf{x}_i$ is the ith sample, and $\mathbf{w}_{ij}$ are the weights of this sample with regard to the jth label. m and n are the total number of samples and labels, respectively. The function is derived by taking the negative of the logarithm of the probability mass function, which is the probability that a sample is exactly equal to one or zero, given that the samples are stochastically independent. Thus, by minimizing this function, the model is trained such that the probability of finding a sample in one of the classes is maximized.

A nonlinear activation function is chosen to capture nonlinear patterns in the data. An activation function often used with the hidden layers is the rectified linear units function (ReLU) [46], defined as

$$\phi(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} \tag{2.12}$$

Because the derivative of the ReLU is one for positive input values, the activation unit avoids the problem of vanishing gradients, which is a problem that plagues deep networks [46].

The sigmoid function, defined in (2.13), is a suitable activation function in the last layer because, for each label, it yields the probability that the label is relevant for a given sample.

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{2.13}$$

The loss function is minimized through gradient descent and backpropagation [46]. Gradient descent is a iterative process for updating the weights in the network by moving in the negative direction of the gradient of the cost function, with respect to the weights. A complete iteration of the gradient descent is referred to as an epoch. The gradient is calculated with backpropagation, where the error in the last layer of the network is propagated backwards through each layer of the network. In batch gradient descent, the weights are then updated using the following rule:

$$\mathbf{w}_k = \mathbf{w}_{k-1} - \eta \nabla J(\mathbf{w}) \tag{2.14}$$

where $\mathbf{w}_k$ is a vector of weights in the kth epoch, η is the learning rate and $\nabla J(\mathbf{w})$ is the gradient of the cost function with respect to the weights. All weights are updated simultaneously after the error of each sample has been evaluated individually. Mini-batch gradient descent is a variation of batch gradient descent where the set of samples are partitioned into smaller batches.

Momentum is concept that can speed up the gradient descent. In addition, momentum filters out small variations in the cost function and can help the gradient descent algorithm from getting stuck in a local minimum. When adding momentum, weights are not only updated based on the gradient of the cost function, but also based on the previous weight changes [47].

The optimizer of the neural network determines how the weights are updated. RMSProp is a type of mini-batch optimizer, and is similar to gradient descent with momentum [54]. The difference is that the optimizer calculates a moving average of the square of the gradient, defined as

$$v(\mathbf{w}, t) = \rho v(\mathbf{w}, t-1) + (1-\rho)\left(\nabla J(\mathbf{w})\right)^2 \tag{2.15}$$

where ρ is a weighting factor. The gradient of the cost function in (2.14) is then divided by the square root of this average.

When training a model, one risk that the model captures the information in the training data set very well, but generalizes badly. This is called overfitting and can be combatted with regularization methods. The model can be regularized by adding dropout layers, where a fraction of the weights in the preceding layer are set to zero during training [9]. This forces the model to learn redundant patterns, which force the model to become more robust. During testing, no weights are zeroed out, but are scaled by a factor equal to the dropout rate to take into account that more activation units are active. Layers can also be regularized by adding an additional cost to weights that are large. One such measure is L2-regularization, where a cost that is proportional to the squared value of the weight is added to each weight [9].

2.4 Electricity Price Forecasting

Power markets differ from other types of markets in that electricity cannot be stored, and supply failures come with large costs. In the day-ahead market, sellers specify how much energy they can deliver, and for which price they are willing to sell. Bids are made by operators of power plants, and bids are made for each price area. Buyers assess how much energy they need to meet demand the next day and the price they are willing to pay each hour. Electricity prices are set where the supply and demand of all the actors are in equilibrium. The goal of the market is to establish equilibrium between supply and demand

Hydrothermal scheduling with mathematical programming has proved efficient in forecasting electricity prices [30], and is also useful in valuating the different physical energy products. Hydro-thermal models, such as the ones used for modelling the Norwegian electricity market, are usually divided into smaller models with various degree of time perspective and level of detail [21, 32].

Long term scheduling has a time horizon of several years, and typically a time increment of one week. A medium term model is often included, serving as a link between the long term and short term models. The most detailed representation of the market and hydrothermal system is given in the short-term hydrothermal scheduling models, and is described in further detail in section 2.4.1.

Several articles have explored the use of machine learning in electricity price forecasting, and a recent overview is given in [45]. The models based on machine learning is only concerned with consumer prices and, unlike models based on system simulation, cannot yield marginal prices on all energy products, such as water values. Furthermore, the machine learning models do not give a complete solution to the scheduling problem in the hydrothermal system. [64] gives an extensive literature overview of long and mid-term electricity price forecasting models, not restricted to the Northern European market, in addition to presenting a new probabilistic model. Machine learning has also been used extensively in energy planning models for forecasting energy demand and consumer load [13].

2.4.1 Short-Term Hydrothermal Scheduling

The goal in short-term hydrothermal scheduling is to establish the most efficient use of hydro and thermal power resources, taking into account the uncertainty in weather, consumer load and prices in exogenous markets [23]. This is done to minimize the cost of operating the entire system, and thus maximizing the welfare of the society. In a de-regulated market, such as the Nordic power market, each producer will in principle try to maximize their own profit.

The hydrothermal systems are comprised of multiple price areas that can contain several water reservoirs, water interconnections and hydro power plants. In addition, thermal power production can be included, as well as other forms of renewable energy production, e.g. wind and solar power. Transfer of electricity between the areas are limited by the capacity of the power lines that connect these areas. The prize zones are usually determined by the connections that limits the transmission capacity [23]. In addition, transmission system operators (TSOs) may impose ramping restrictions [44], limiting the change in power flow per time unit. In addition to hydro power, thermal power plants, or other forms of power production from renewable energy sources, e.g. wind and solar power, can be present.

Several different approaches have been used to solve short-term hydrothermal scheduling problems. The different types of methods can broadly be divided into two categories: deterministic methods and heuristic methods [20]. Linear programming and mixed-integer programming are among the deterministic methods that have been applied, whereas evolutionary computing and artificial intelligence are examples of heuristic methods. One advantage of the heuristic approaches are the stochastic randomness of the methods, which helps in avoiding that the model get stuck in local optima. Another advantage is that the methods can be applied on several types of problems, regardless of whether the objective function and constraints are linear or not. With deterministic models, there are different models for different types of problems. On the other hand, deterministic models have the advantage of the availability of efficient solvers and

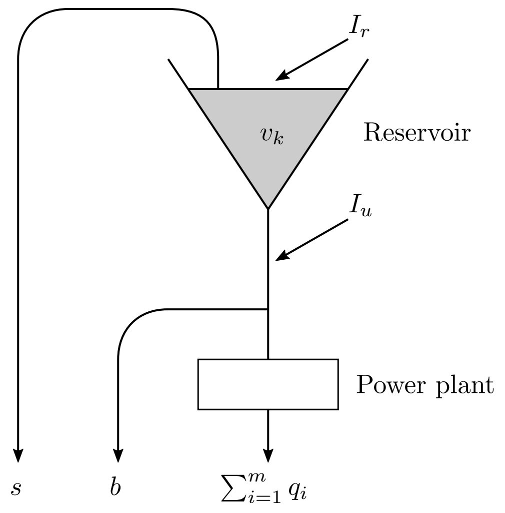


Figure 2.1: The figure shows all the components of a module in the hydropower system. The figure shows regulated inflow (I_r), unregulated inflow (I_u), spillage (s), bypass (b) and discharge ($\sum_{i=1}^{m} q_i$). The reservoir level at timestep k is denoted v_k.

a robust and well studied mathematical framework.

2.4.2 PRIBAS Project

At SINTEF Energy Research, a new hydrothermal electricity price forecasting model is developed with a higher level of detail, and a first version is presented in [32]. The model consists of a long-term strategic model which is coupled with the short-term operational model named PriMod.

The strategic model [33] is stochastic and creates a fan of scenarios based on uncertainties in electricity loads and weather. The possible scenarios are given in the form of Bender's cuts, which are then used as input data in a more detailed short term operational model. The cuts are the expected future costs as a function of the reservoir levels.

The hydrothermal system is modelled as a set of interconnected price zones and exogenous markets. Each price zone consists of a network of modules that each consist of one reservoir. The reservoirs are connected to the rest of the system through three waterways: inflow, release and spillage. Release is regulated water flow from the reservoir, whereas spillage occurs when the reservoir is overflowing. Within the module, there can be a hydro power plant attached downstream from the reservoir. When a hydro power plant is present in the module, water released from the reservoir, together with unregulated, non-storable inflow, either bypasses or is dispatched through the plant. A module, including the reservoir, power plant and all the waterways, is illustrated in figure 2.1.

The model is formulated as a deterministic mixed integer programming (MIP) problem, where the cost function to be minimized represents the total system cost. The operational model solves weekly problems that are based on physical input data and the Bender's cuts from the long-term strategic model. Water and power balances are implemented as constraints in the problem formulation. The linear equations are arranged such that the right-hand side can give physical and economic insight through the calculation of the corresponding dual values.

For each module, the reservoir head, inflow, release and spillage must be balanced in each time step of the simulation. Water dispatched to the power plant is divided into segments for approximating the water turbine efficiency curves as a function of water head.

In each price area and exogenous markets, power generation and load must be balanced, taking account of transmission losses and generation inefficiencies. Other constraints, e.g. related to start-up costs of power plants or ramping restrictions in the transmission network, are implemented into the model [32].

2.4.3 Time Resolution and Computational Cost

The computational performance of several methods for solving short term hydrothermal coordination problems are discussed in [20]. The problem of computational time with MIP is well known, especially with large problems such as short term hydrothermal scheduling problems.

To overcome the higher computational costs of a more detailed system description, the size of the operational problems can be reduced by splitting the problems into smaller subproblems. One way of achieving this is through interpolating the function for the expected future cost [29]. Splitting a weekly deterministic problem into smaller daily problems adds uncertainty to the model which mirrors the increased uncertainty in power generation from variable renewable energy resources.

Another method for tackling the increased computation costs, is presented in [48]. The authors have investigated the error from reducing the time resolution in electricity market models, and present a method to identify which time steps can aggregated. Furthermore, the authors have presented an algorithm that makes it possible to rank the influence of each time step aggregation measure.

The potential of employing parallel processing to decrease computation times is explored in [57], and models such as PriMod can benefit from parallelization.

Chapter 3

Method

3.1 Case

The main dataset consist of 19 price areas with hydropower production. In total, there are 1100 modules divided over these areas. Wind power is produced in 13 of these areas, and the system is connected to eight exogenous markets with thermal power production.

3.2 Solving the Linear Programming Problems

Before solving the problems, the models formulated as mixed integer programming problems were converted to linear programming problems by fixing the binary variables. IBM ILOG CPLEX version 12.7.1.0[1] (referred to as CPLEX from now) was used to solve the resulting LP problems. Settings were set and commands were executed through an interactive solver. The solver returns dual values and reduced costs together with the solution of the problem. To optimize the problem, CPLEX was set to employ the dual simplex method. Using the same solution method each time is necessary for getting comparable results, and to avoid that the solver spend time on finding the optimal solution method for the problem.

CPLEX has built-in pre-processing that can eliminate rows and columns from the problem [36]. The pre-processing was left on so that any improvements due to the reduction in size of the problem, come in addition to the optimization done by the pre-processing. It was also ensured that the solver did not use an advanced basis, which would mean that the solver use the previous optimal basis as a starting point when solving the next. All other CPLEX parameters were left at their default values.

For measuring the computational performance, CPLEX returns the solution time in seconds and in deterministic ticks [36]. The solution time in seconds is dependent on the processor

[1]`https://www.ibm.com/products/ilog-cplex-optimization-studio`

and other hardware, as well as the operating system and processes running in the background. Deterministic ticks, however, are independent of these factors and give the same results each time the model is solved. Thus, deterministic ticks have been used to measure the relative changes in solution times. The number of iterations made by the solver is also measured, which give an indication on the performance. However, the time spent on each iteration will vary, such that a reduction in the number of iteration will not necessarily lead to lower solution times.

3.3 Removing Constraints

In a general LP problem formulated as (2.1), the set of feasible solutions are constrained by the equations that form (2.1b). Figure 3.1 illustrates an LP with two variables x_1 and x_2, both restricted to be positive, and five restrictions $c_1, c_2, \ldots, c_5$ represented by solid lines. The grey area is the resulting region of feasible solutions. When letting either of the two variables be constant, the cost function can be represented by a straight line, as illustrated by the dashed lines in the figure.

As mentioned in section 2.1.1, the optimal solution will be an extreme point of the convex solution set for (2.1). Thus, for two variables, the optimal solution will be at an intersection of two constraints, indicated by the circles in the example shown in figure 3.1. The figure shows that the optimal value must be found at one of the edges of the feasible region. Both the simplex and the dual simplex algorithms iterate through a subset of the edges of the feasible region, until it is determined that the optimal solution is found. With several variables, the two-dimensional intersecting lines are replaced with intersecting hyperplanes, but the general principle of the simplex algorithm stays the same.

By removing constraints that do not contribute to the objective value, the number of intersections will be reduced. As a result, the number of iterations needed to reach the optimal solution is potentially reduced as well. In figure 3.1, for example, removing constraint c_1, c_2 and/or c_5 will not change the value of the optimal solution, and there is potentially one less basic feasible solution to examine in the simplex procedure.

After successfully solving the linear programming problem, the dual values were examined in order to find constraints that did not contribute to the overall cost of the system. As established in section 2.1.3, all constraints with a dual value of zero can safely be removed without altering the solution, unless the removal of the constraints causes numeric instability, e.g. by causing a high degree of degeneracy. These constraints were then removed from the original problem, before the modified problem was solved to find the change in computation time, and to verify that the solution did not change due to the modification.

The number of constraints in the linear programming problems ranged between roughly 4000 and

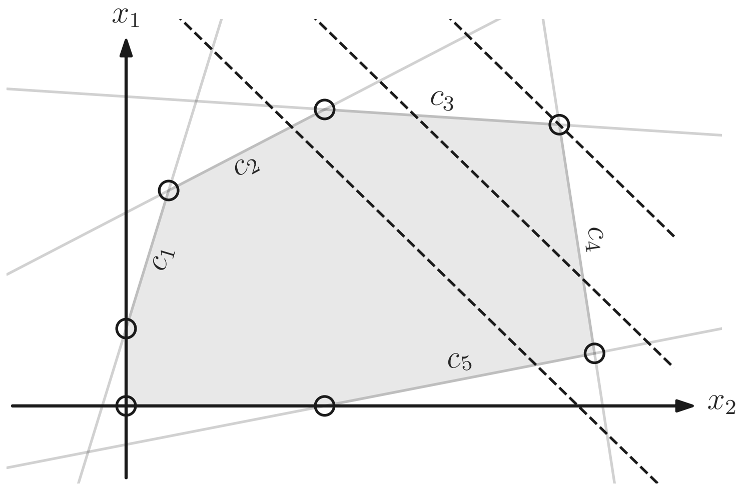


Figure 3.1: The figure represents a linear program with two variables x_1 and x_2 and marks the region of feasible solutions constrained by five different restrictions. The cost function $\mathbf{c}^T\mathbf{x}$ for three different feasible solutions $\mathbf{x} = (x_1, x_2)$, are illustrated by the dashed lines.

250,000. In addition, there were between 85,000 and 380,000 bounds. The amount of constraints and bounds made it necessary to automate the process of solving the problem, identifying the constraints and bounds that could be removed, modifying the original problem, before solving the problem again. The automation was done with a script written in Python. The solution files returned by CPLEX use the XML[2] file format, and Python packages for parsing such files are readily available. However, for reading the files containing the problem definition and used as input for CPLEX, a parser was written from scratch. The parser use regular expressions to extract and identify information.

3.4 Combining Water Balances

Each time step consists of a water balance for each module, that is implemented as a constraint in the linear program. For a single module, the constraints for all 24 timesteps are formulated as

$$v_1 + \sum_{i=1}^{m} q_{1i} + b_1 + s_1 = v_0 + I_1 \tag{3.1a}$$

$$v_k - v_{k-1} + \sum_{i=1}^{m} q_{ki} + b_k + s_k = I_k, \quad k = 2, 3, \dots, 24 \tag{3.1b}$$

where v_k, b_k, s_k and I_k represent the water level, bypass, spillage and inflow, respectively, of a particular module in time step k. The reservoir level at the start of the daily problem is v_0, and

[2]Extensible Markup Language (`https://www.w3.org/XML/`)

the ith discharge segment for time step k is denoted q_{ki}. The equation for the first time step is separated from the rest because v_0 and I_1 is combined into a common input value, whereas in the subsequent timesteps the variable for the water volume of the previous timestep is included, and only the inflow I_k is used as input.

If the water volume of a single reservoir does not reach the upper or lower water limits in several consecutive time steps during the daily problem, the water balances of these time steps can be combined. The intermediate values for the water volume will not have an impact on the total cost of the system, because any power plant connected to the reservoir will have water available for producing power at maximum capacity if desired. The water that leaves the reservoir, except an amount that may be required to bypass the plant, can be used for electricity production. Furthermore, there will be no spillage, and no water must be released to avoid exceeding the reservoir limit. Thus, no potential energy will be lost, and there is no reduction in potential income from hydro power production. As a result, the water balances of these consecutive time steps can be combined such that the intermediate water levels, i.e. all the water levels except the first and the last, will cancel. In addition, only one spillage variable is needed, as no spillage can occur in any of the combined time steps except the last. If the water balances for the n first time steps are combined, the resulting constraint can be expressed as

$$v_n + \sum_{k=1}^{n}\left(\sum_{i=1}^{m} q_{ki} + b_k\right) + s = v_0 + \sum_{k=1}^{n} I_k \tag{3.2}$$

which is the sum of the water balances in (3.1) from the first to the nth time step.

Before combining the water balances, it must be determined if and when the water limits of the reservoir is reached. The water levels are only known at the beginning and end of the day. For the remaining time steps, the water volume is governed by the inflow and release during that time step, together with the water volume of the previous time step. The highest possible water volume is found by taking the highest possible water volume of the previous time step, adding the inflow, then subtracting the minimum bypass and dispatch flows. Thus,

$$v_k^{\text{high}} = v_{k-1}^{\text{high}} + I_k - \sum_{i=1}^{m} q_{ki}^{\text{min}} - b_k^{\text{min}} \tag{3.3}$$

where v_k^{high} is the highest possible water volume for time step k, provided that the water volume remains within the reservoir limits. I_k is the inflow and b_k^{min} is the minimum allowable bypass flow. The minimum value of discharge segment i is q_{ki}^{min}. Similarly, the minimum possible water volume is found by adding the inflow, and subtracting the minimum bypass and maximum

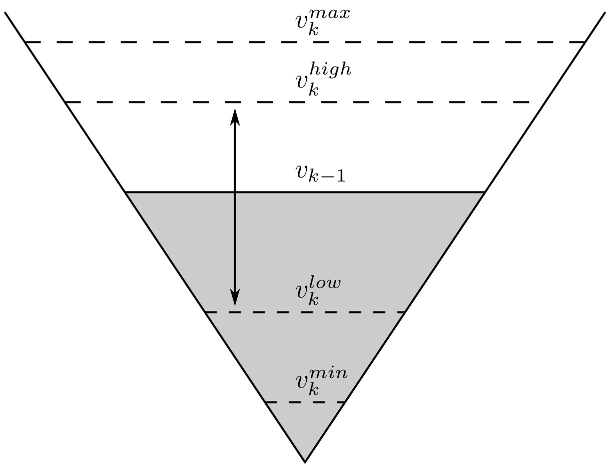


Figure 3.2: Possible change in reservoir level from one time step to the next. The reservoir level of the previous time step is denoted v_{k-1}. The maximum and minimum reservoir levels for this reservoir are denoted v_k^{max} and v_k^{min}, respectively. v_k^{high} and v_k^{low} are highest and lowest possible water levels, respectively, in the current time step.

dispatch from the lowest possible water volume of the previous time step:

$$v_k^{\text{low}} = v_{k-1}^{\text{low}} + I_k - \sum_{i=1}^{m} q_{ki}^{\text{max}} - b_k^{\text{min}} \tag{3.4}$$

where v_k^{low} is the lowest possible water volume for time step k, and q_{ki}^{max} is the minimum value of discharge segment i.

An algorithm was created to check if the some of the first water balances for each module in a daily problem could be combined. In the first time step for each module, the highest and lowest possible water volumes can be compared with the maximum and minimum allowable water volume, respectively. For each subsequent time step k, equations (3.3) and (3.4) are evaluated against the reservoir limits. An illustration of this is given in figure 3.2, which shows the possible variation in water head in a given time step together with the reservoir limits. The algorithm continues as long as the reservoir limits are not exceeded by the highest or lowest possible water volume, and terminates when the limits are exceeded or when the all the time steps of the day have been iterated over. When the algorithm terminates, the water balances up to the current time step are combined using equation (3.2). For the remaining time steps of the day, it is unknown whether the water volume has reached the reservoir limits, and no further steps are made by the algorithm. An extension of the script used to remove constraints and bounds in section 3.3 was used to modify the linear program files used by CPLEX. All bounds for the water volume variables that cancels are removed. The same is done with the bounds for the spillage variables that have previously been removed from the water balances.

3.5 Problem Size Reduction with Genetic Algorithms

Simply removing a row from the linear programming problem can cause the problem to become infeasible. Instead, the constraints are deactivated by adding a free variable to the constraint. Given a system of constraints $A\mathbf{x} = \mathbf{b}$, a modified system that corresponds to an individual in the genetic algorithm can be expressed as

$$A\mathbf{x} + I\mathbf{y} = \mathbf{b} \tag{3.5}$$

where $\mathbf{y} \in \{0, 1\}^m$ is a binary vector representing an individual in the genetic algorithm, and I is the $m \times m$ identity matrix. The binary values are labels that indicate whether a constraint in the LP problem can be deactivated or not. Ones indicate that a constraint is included and zeros that a constraint can be removed.

The genetic algorithm was implemented using DEAP, which is a framework written in python for performing evolutionary computations [22]. Genetic algorithms and multi-objective optimization are among the tools that are provided by DEAP.

The (μ, λ) evolutionary strategy was used in the genetic algorithm. The number of individuals to select for the next generation (μ) was set to 100, and the number of children to produce each generation (λ) was set to seven times the value of μ, as experimental studies have shown that approximately this ratio is optimal [2]. The algorithm used two-point crossover, with the probability of producing an offspring due to crossover set at 70%. Mutation was done by randomly flipping bits in the vector representation of the individuals, where the probability of a bit being flipped was five percent. NSGA-II was used for the selection process. In some cases, the deactivation of a constraint makes the problem infeasible. When this happens, the objective functions are evaluated to a value a few order of magnitudes larger than otherwise expected, which penalizes these individuals and reduces the likelihood that these individuals are carried over to the next generation.

The script implementing the genetic algorithm with DEAP is found in appendix B.1.

3.6 Constraints Classification

One challenge in predicting the constraints that can be removed is the large number of individual constraints that must be classified. The scheduling problems that are looked at in this thesis have several million constraints. Creating a multiclass multi-label model, with each label representing a constraint, for such a vast number of labels is computationally and statistically challenging. To overcome this problem, the problem has been transformed by splitting the label space into smaller parts, creating classification models that each focus on specific types of constraints in the hydrothermal scheduling model. Two advantages of this approach are the computational benefit and the interpretability of the results. The main drawback of this method is that relations between different parts of the original model may not be discovered by the machine learning model. Unlike in the binary relevance method, however, relationships within each partition may still be facilitated.

In the present work, and as a first step, several simplifications are made to handle the vast amount of data. First, the LP model formulation is simplified by assuming that import and export of power from exogenous markets stay constant, and that restrictions on bypass and reservoir levels are not subject to seasonal changes. Thus, only the right hand sides of the water balances and the power balances are varied, and none of the bounds are changed. The reservoir levels stay the same at the start of each week but are allowed to change during the week. When training the model, only variations in inflow, load and wind power are considered.

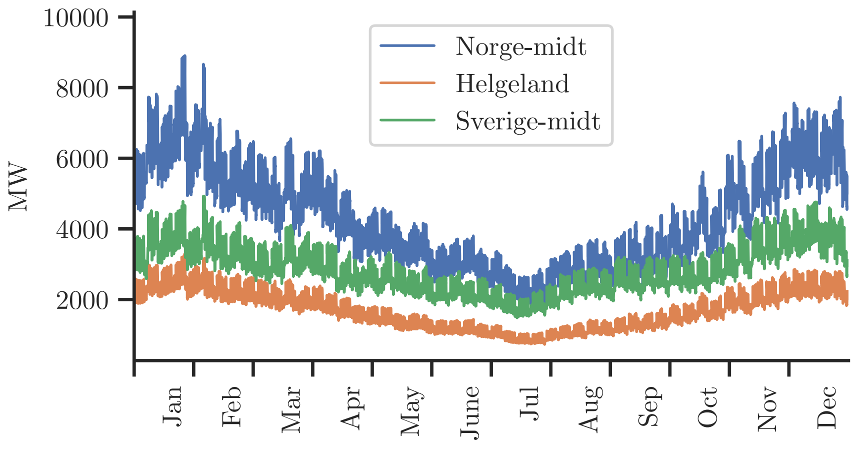
The genetic algorithm needs much time and resources to optimize the number of constraints. To assess the potential of the methodology, the constraints are only deactivated based on their dual values in the optimal solution. This was due to the lack of computing resources to create a sufficient amount of target data with the genetic algorithm.

The Hamming loss, multi-label accuracy and F1-score, given by equations (2.7)-(2.9), are measured. The baseline binary score is set to be the score that would be achieved if all the labels that equal one in more than 50% of the samples are predicted to always equal one, and the remaining labels are predicted to always equal zero. The main goal is to find the accuracy of the reduced linear programs suggested by the machine learning model, and the change in computational times. This is measured with the mean absolute error and percent deviation, respectively.

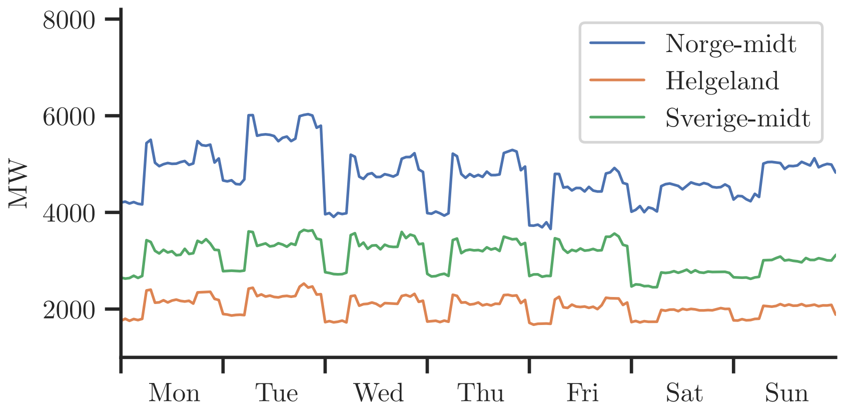
3.6.1 Input Data

Load

The load profiles are artificially generated data series based on common power consumption pattern, which are then adjusted with historical temperature data. The temperature corrections are based on data from the period 1961–1991. Gaussian noise with 1% standard deviation have been added to the data. This was to avoid several repeated values and to have a more realistic profile. Moreover, the noise may benefit the training of the machine learning models, and ultimately create more robust models. A typical load profile for one year is shown in figure 3.3a, and a closer view of the load profile during a week is shown in figure 3.3b. The load is lower during the summer with less need for electrical heating. The load is also lower during the weekends, as commercial buildings and industry typically consume less power during this period. The data has an hourly resolution.



(a) One year load profile.



(b) One week load profile.

Figure 3.3: The two figures show load profiles in three different areas. The first figure (a) shows the graphs for one year, and the second figure (b) graphs for one week in March.

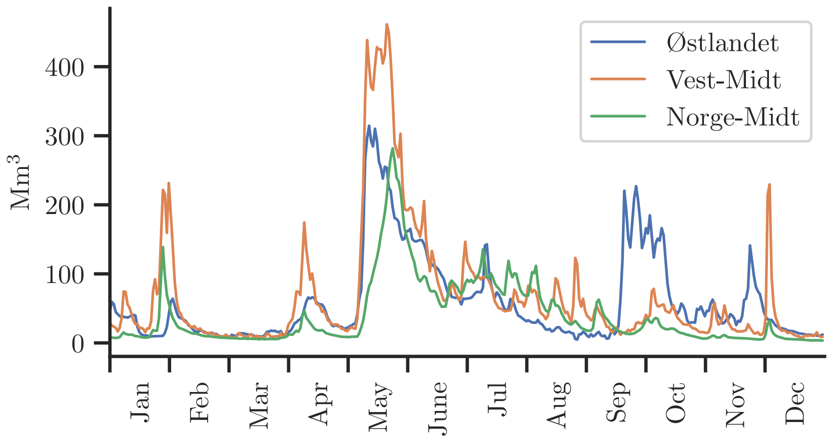


Figure 3.4: Total inflow in three different price areas. The curves show the sum of the amount of water that flows into one area per hour.

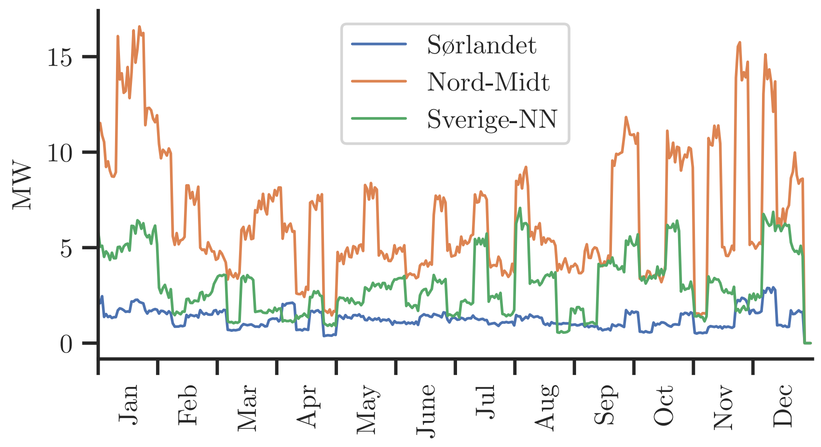


Figure 3.5: Wind production in three different areas.

Inflow

Inflow data is based on historical data from the same period as the temperature corrections for the load profiles. The data has a daily resolution. The accumulated inflow during a year to all reservoirs is shown in figure 3.4 for three different price areas. The units of the inflow data is Mm^3. The inflow is relatively low during the winter months, and peaks during the spring, with large amounts of meltwater.

Wind Power

The wind power generation data does also have daily resolution. The same type of Gaussian noise that was added to the load data is added to the wind power data, for the same reason as with the load data. As can be seen in figure 3.5, wind power generation is erratic throughout the year.

3.6.2 Target Data and Class Imbalance

The targets used during training are obtained by solving the LP problems and extracting the dual values in the optimal solution.

The targets are represented by a binary vector $\mathbf{y} \in \{0, 1\}^m$, as in section 3.5, where each value is a label that corresponds to a constraint in the model. In the remainder of the thesis, labels are said to be relevant or present when the corresponding constraint should remain in the LP model, which is represented by the binary value one in the vector $\mathbf{y}$.

Several labels always have the same value. That is, the count of samples associated with a particular label is either zero or equals the number of training samples. These labels are removed from the machine learning model. When making predictions for the training samples, the predicted value for these labels are always zero or one.

Labels that are present in more than 99% of the samples were also excluded because it is hard to beat a baseline of 99% prediction accuracy, and there will be very few training and validation samples. Furthermore, there is little to gain in computational times by leaving these labels in the model, as the corresponding constraints rarely are removed from the LP model. Labels that are relevant in fewer than 1% of the samples were also excluded. Once again the baseline binary prediction accuracy would be 99% (by setting all samples to zero), and there would be very few samples for training the model. In the latter case, even though the corresponding constraints rarely contribute to the total cost in the LP model, they are always set to be included because of the challenge in training a model to learn when they should be included or not, and to make sure that the omission of these constraints do not detriment the accuracy of the reduced LP problem.

The frquency of the remaining labels that correspond to the water balances are present, are shown in figure 3.6. The figure shows that the distribution of label frequencies is heavily imbalanced. The label imbalance must be tackled differently in multi-label multiclass problems than in single-label multiclass problems.

One way of dealing with the imbalance is by assigning different weights to the labels. Labels that occur often are given smaller weights than labels that occur seldom to make the model take more notice to the minority labels. Little information in the literature on how to assign such weights was found. In the present work, the weight of a label is assigned by subtracting the count of samples belonging to this label from the total number of samples, and then normalizing these values. In other words, letting $\mathbf{y}_i = (y_{i1}, y_{i2}, \dots, y_{im}) \in \{0,1\}^m$ be the target vector for sample i, as in section 3.5, the weight w_j of the jth label is

$$w_j = 1 - \frac{1}{m}\sum_{i=1}^{n} y_{ij} \tag{3.6}$$

where m is the number of samples and n is the number of labels.

The mean imbalance ratio per label, obtained by averaging the value of (2.10) for all labels, is listed for the six label divisions in table 3.1. The number of labels together with the mean imbalance label ratio give an indication on the difficulty of creating a well performing model.

Another issue, closely related to the imbalance problem, is the distribution of label frequencies in the train-test split. In classification problems, a simple way of assigning samples to each fold is by shuffling the samples before splitting the data. In this method, however, the labels may not be equally represented in the different folds. In the worst case scenario, some labels may not be represented in one of the folds. An alternative is to use stratified sampling during the splitting to ensure that there is a balanced representation of each label in each fold. This form of stratification, however, becomes impossible when the number of unique label combinations

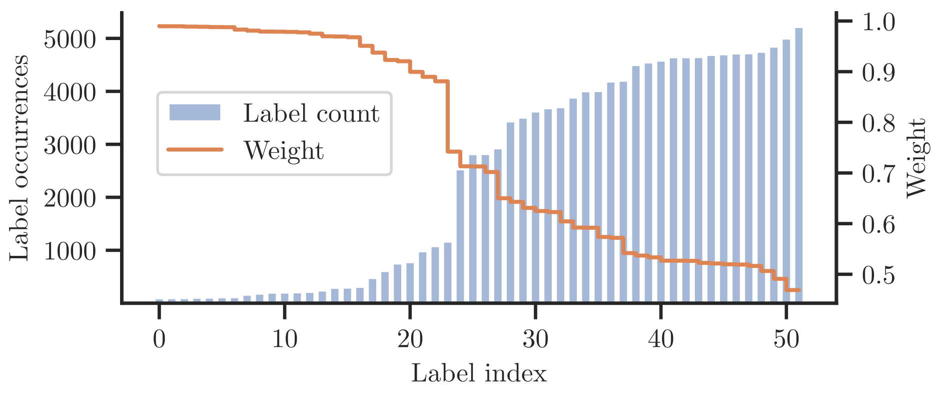


Figure 3.6: Illustration of the target imbalance and the target weights for the labels associated with cable ramping. The occurence of each label is plotted and the primary axis indicates the index of the label. The labels are indexed in order of how often they are relevant.

Table 3.1: The mean imbalance ratios (mean of equation (2.10) for all labels) for each of the the six label divisions.

Water balances	3.50
Discharge balances	4.35
Cable ramping constraints	12.5
Reservoir bounds	32.5
Bypass bounds	16.5
Discharge limits	10.5

Table 3.2: Summary of input data. Number of features in the different inputs.

Inflow	1022
Wind power	13
Load	432
Total	1467

Table 3.3: The distribution of training, validation and test samples. The numbers of training and validation samples are approximate due to the nature of the iterative stratification method.

Training samples	≈ 7371	$\approx 67.5\%$
Validation samples	≈ 2457	$\approx 22.5\%$
Test samples	1092	10%

(2^m, where m is the number of labels) far exceed the number of samples. An alternative way of assigning labels to the different folds in multi-label classification is called iterative stratification and is presented in [49]. It is shown that iterative stratification performs better than random shuffling when the number of distinct label sets is much higher than the number of samples. Iterative stratification works by calculating the desired number of samples for each label in each fold. The algorithm then iterates through each label, starting with the label that is represented in the smallest number of samples. Then, for each sample with this label, the samples are assigned to the fold where the number of desired samples for a sample with this label is highest. At the end of each iteration, the numbers of desired samples are recalculated.

The data was split into a training and a test set, randomly shuffling the data, with 10% of the data used as test data. Using k-fold iterative stratification to preserve a better balance between the labels would be ideal. However, the size of the data set, having 10920 rows and more than 240,000 columns, made this method computationally challenging with the computer hardware at disposal.

The training set was further split into four folds for cross validation during training. A summary of the number of labels and features are given in table 3.2. For the training set, k-fold iterative stratification could be used. This was possible after splitting the training data into smaller sets corresponding to each of the constraint types, and removing constraints that always or never had dual values of zero. The train-test split, on the other hand, needed to include all samples because the models for each constraint were to be used on the same test set. The number and distribution of samples between the training, validation and test sets is given in table 3.3. The number of test samples is exact, but the number of training and validation samples are approximate because the iterative stratification approach can deviate slightly from the desired number of samples in each fold.

3.6.3 Artificial Neural Network

The input data has either hourly or daily resolution. When the resolution is hourly, there are 24 input values for each hour of the day that correspond to only one constraint. The data of one sample with hourly resolution is therefore represented in a two dimensional matrix, where one dimension is the number of time steps, and the other dimension separates each area. As a consequence, the model is fed with two different inputs of different shapes. The two inputs are first treated separately. The multidimensional data is fed into a time distributed layer with twice the amount of output channels as input channels. The time distributed layers applies a densely connected network on all the channels in the feature matrix, but only using the number of activation units as in one of the channels. After this, the feature matrix is flattened and concatenated with data originating from the other input. The second input is fed into a densely connected layer where, once again, the data is oversampled by having twice as many activation units as there are features. The layers are oversampling the input to ensure that the model can capture more details of the underlying data.

The concatenated feature matrices are then fed into several hidden densely connected layers. The final layer is fully connected and has a sigmoid output. The size of the final layer matches the number of labels that are to be predicted.

Each densely connected layer are L2-regularized. The model is further regularized by separating each densely connected layer with a dropout layer, having a dropout rate of 0.5. Overfitting is dealt with through the regularization and dropout layers rather than creating smaller or fewer layers.

RMSProp was used as the optimizer, with a learning rate (η) that was varied between 1×10^{-5} and 5×10^{-5} for the various models. The moving average parameter (ρ) is left at the default value of 0.9 in both cases.

The number of parameters for each model is shown in table 3.4. The same model structure is used for each model, but the number of activation units in the two final layers are scaled to match the number of labels that are to be predicted. The second last densely connected layer has $(1467 + 1) \times k$ parameters and the final layer has $(k + 1) \times k$ parameters, thus the number of parameters increase quadratically with the number of labels. Therefore, the number of nodes in the last hidden layer would likely need to be decreased because of computer memory limitations if the number of labels grows larger than what is seen in the case that is studied here.

The model is implemented with Keras and TensorFlow, and the code is shown in appendix B.2.2. Keras evaluates and logs the binary accuracy, but no suitable evaluation metrics for multi-label classification are predefined in Keras. It is possible to define custom metrics, however. The Hamming loss, multi-label accuracy and F1-score are implemented as custom metrics using the

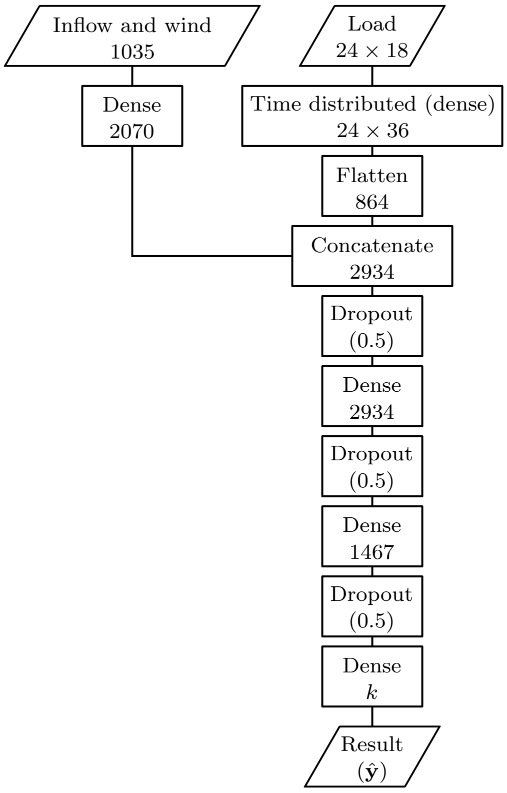


Figure 3.7: Structure of the artificial neural network, show all the hidden layers in addition to the input and output layers. The number below each label in the diagram indicates the number of output parameters of that layer.

Keras application programming interface (API), and the code is included in appendix B.2.1.

The features are standardized before being fed to the model, using the following formula:

$$x'_j = \frac{x_j - \bar{x}}{\sigma} \tag{3.7}$$

where x'_j is the standardized value of x_j, which is the feature value for the jth sample. $\bar{x}$ is the mean across all samples and σ is the standard deviation. The performance of the RMSProp optimizer benefits from standardization [46], and it ensures that the errors to be minimized are on the same scale. Care is taken to use the mean and standard deviation of the training features when standardizing the validation and test features, so that the validation or test features are brought to the same scale as the training features.

The script for performing cross validation, training and test predictions is attached in appendix B.2.3.

Table 3.4: The table shows the number of labels and the total number of parameters for the model for each constraint.

Constraint	Labels	Parameters
Water Balances	716	3.71×10^6
Discharge Balances	527	3.20×10^6
Cable Ramping Limits	51	2.22×10^6
Reservoir Bounds	52	2.22×10^6
Bypass Bounds	6613	55.6×10^6
Discharge Limits	1419	6.24×10^6

3.7 Framework for Reducing the Problem Size

The results of analyzing the dual values and the results of the genetic algorithm can be used to create training targets that indicate which constraints can be removed. These targets can the be used to train a classifier that can be implemented into the existing hydrothermal scheduling model.

3.7.1 Training

The targets used for training the machine learning models are binary vectors indicating whether a constraint is deactivated or not. The targets can be computed by first solving the linear programming problems, then evaluating the dual values to find which constraints can be removed. Next, an evolutionary algorithm, like the genetic algorithm described in section 3.5, can be used to reduce the problem size even further. Removing constraints with a dual value of zero before using the genetic algorithm can significantly reduce the size of the population that must be evaluated by the genetic algorithm. Using targets produced by these two methods applied on several hydrothermal scheduling problems, a machine learning model can be trained to find patterns that can tell when constraints can be deactivated. Target data for training is obtained through the process illustrated in figure 3.8.

Combining water balances before creating a target vector creates difficulties when used together with a machine learning model. In contrast to the removal or deactivation of constraints, combination of water balances modifies the constraints and the total number of constraints.

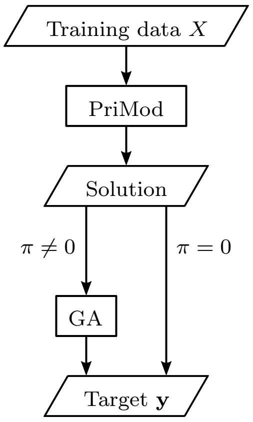


Figure 3.8: Flowchart showing the process of generating target data for use in machine learning. In this figure, π denotes the dual value of a constraint in the solution file.

3.7.2 Implementing the Model

The implementation of the machine learning model into the linear programming model is illustrated in figure 3.9. The same input data is fed to both the linear programming model and the machine learning model. The machine learning model identifies constraints that can be deactivated independently of the LP model. Based on the information gained from the machine learning model, the LP problem is reduced before being fed to the hydrothermal scheduling model.

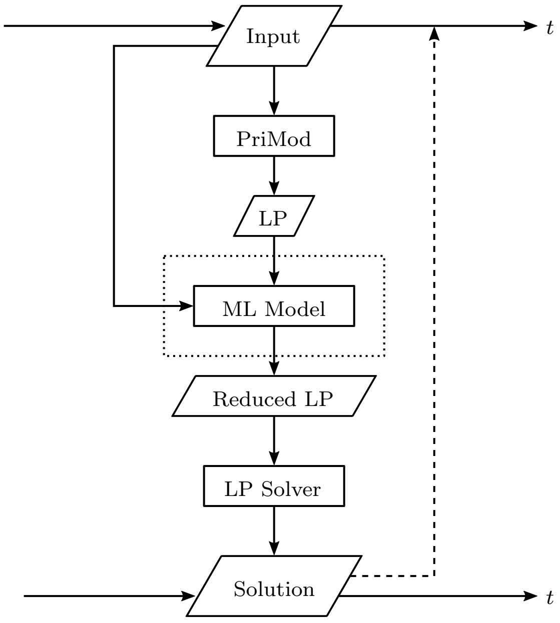


Figure 3.9: This flow chart illustrates how the machine learning model can be implemented into an electricity price forecasting or hydrothermal scheduling model.

Chapter 4

Results

4.1 Removing Constraints

4.1.1 Removal Based on Dual Values

Removing all constraints yielding a dual value of zero consistently results in fewer iterations and shorter solution times. The decrease in computation times and number of iterations is shown in figure 4.1.

The potential in reducing the number of constraints is evaluated for five different linear programming problems. Some characteristics of the LP problems are listed in table 4.1. The LP problems are based on three different scheduling models developed at SINTEF Energy Research: Samnett [34], SOVN [58] and PriMod of the PRIBAS project. SOVN and PriMod are both used with two different datasets, leading to the different number of constraints.

The number of constrains that can be removed varies significantly from case to case. With the LP problems from Samnett and SOVN IG, little is gained by removing constraints with a dual value of zero. In the remaining three cases, however, a large fraction of constraints can be removed. Using the PriMod model and HydroCen dataset, as many as 93% of the restrictions can be removed, which leads to a 35% reduction in solution time. The majority of the constraints that could be removed are constraints placing bounds on individual variables.

4.2 Impact of External Data

Figure 4.2 shows one year of input data together with a count of how often during a day that the cable ramping restrictions places a constraint on the cost formation. There is a correlation between the load and the number of times that cable ramping impacts the cost. In periods with little load, the ramping restriction are not needed. A degree of negative correlation can also be seen between inflow values and the number of cable ramping constraints.

Table 4.1: The number of constraints in the different linear programming problems, together with the number of iterations and computation times in deterministic ticks needed for finding the optimal solutions.

LP problem	Constraints	Iterations	Ticks
Samnett	4170	3793	125.36
SOVN IG	6415	28493	1278.7
SOVN SK	56394	91919	13553
PriMod CES	250910	13511	1053.5
PriMod HydroCen	79164	12476	663.19

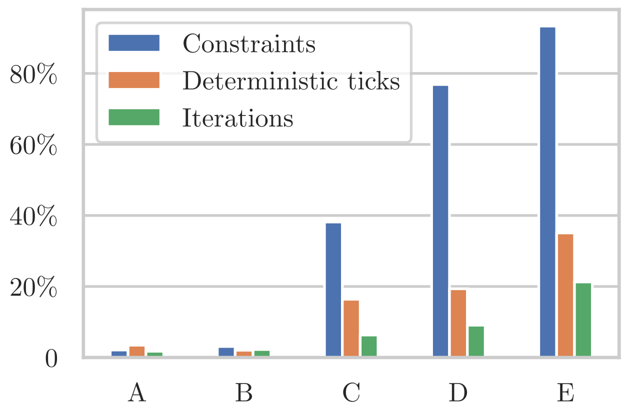


Figure 4.1: The percentwise reductions in the number of constraints (left columns), deterministic solution time (center columns) and number of iterations needed for solving the reduced problem (right columns), are shown for five different linear programs: SOVN SK (A), SOVN IG (B), Samnett (C), PriMod HydroCen (D) and PriMod CES (E).

In July and August, the ramping restriction does not play a role in the modelled power distribution. These months are periods where less power need to be imported to the price area because of the high availability of energy reserves in the reservoirs. There is also less demand in neighboring regions, and neighboring price areas with hydro power are likely to have energy reserves in their reservoirs as well. Moreover, the spikes in load levels are smaller. The variation in load during in these two months is less than half of that in the winter months.

4.3 Combining Reservoir Balances

Because of the hourly resolution, there are 24 water balances like (3.1) for each reservoir in each daily optimization problem. Using the algorithm for combining water balances, the number of water balances was reduced by 64%, which amounts to a 6.8% reduction in the total number of constraints. This led to a negligible 1.5×10^{-9} relative change in the optimal objective value.

For the majority of the reservoirs, several of the water balances could be combined: For 62% of the reservoirs, all 24 water balances were combined into a single restriction, whereas 15% of the reservoirs had 2–23 water balances combined.

The algorithm had biggest impact on the larger reservoirs. Figure 4.3 shows that for most of the reservoirs bigger than $10\,\mathrm{Mm}^3$, all water balances in the daily problem could be merged into one constraint The same figure also shows that most of the remaining reservoirs in this size range had 2–23 balances combined. For most reservoirs (94%) smaller than $0.10\,\mathrm{Mm}^3$, on the other hand, none of the water balances could be combined.

4.4 Genetic Algorithm

When the number of constraints is m, there are 2^m combinations to explore. With $m = 1.6 \times 10^4$, exploring all constraints is computationally infeasible. However, the genetic algorithm attempts to focus the search on the most fit individuals. In the following case, the cost was evaluated for 2.6×10^4 individuals.

A Pareto front is shown in figure 4.4. The Pareto front flattens when the number of constraints are reduced to a little less than 4000, indicating that there is little to gain by trying to reduce the number of constraints beyond this. The total system cost in the base case is 2.7×10^9, while the deviation in cost, corresponding to the Pareto optimal individuals shown in the figure, are eight to nine orders of magnitude lower.

In the discussion below, the base case refers to the LP problem where all constraints having a dual value of zero have been removed. Choosing a threshold deviation of €20,000, indicated by the dashed line to the right in figure 4.4, the other objective function, which concerns the number of iterations needed to solve the problem, has an optimal value of 5680, which is 35%

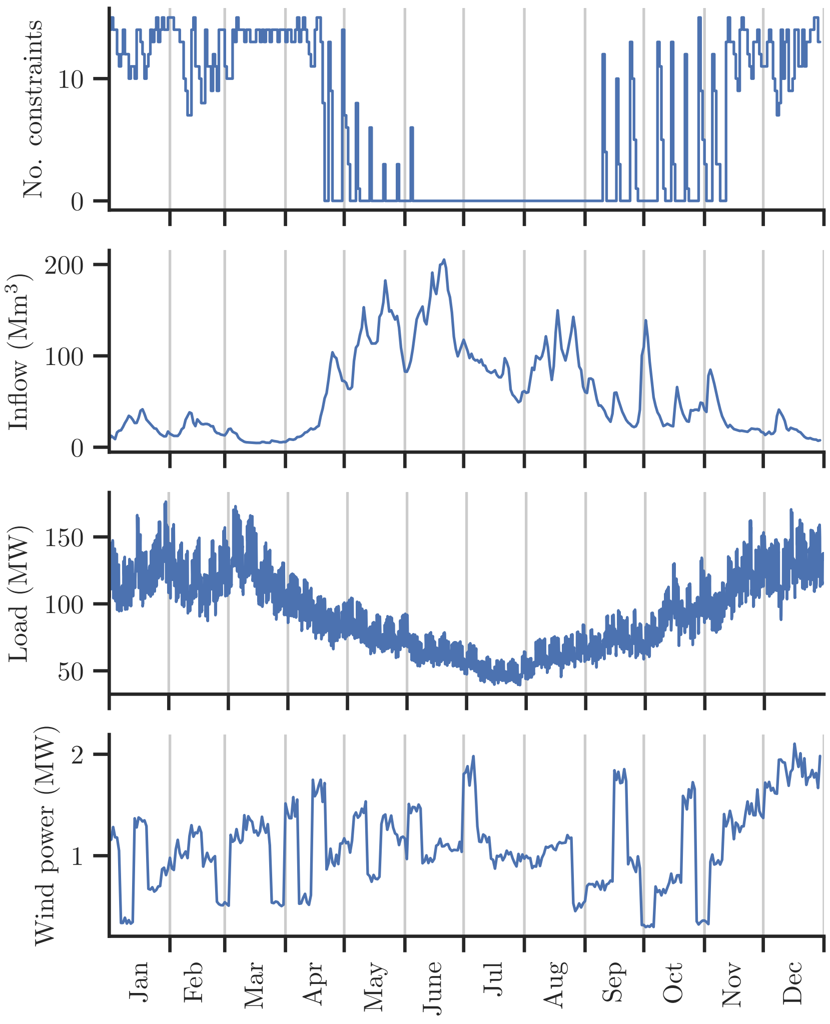


Figure 4.2: The above figures shows the relationship between the yearly variations in inflow, load and wind power production, and the impact theses values have on the number of cable ramping constraints with non-zero dual values in the region “Sørlandet”. The plot at the top shows the number of constraints during each day, the second plot shows the total inflow to the region, the third plot shows the load profile in the region, and the bottom plot shows the wind power production.

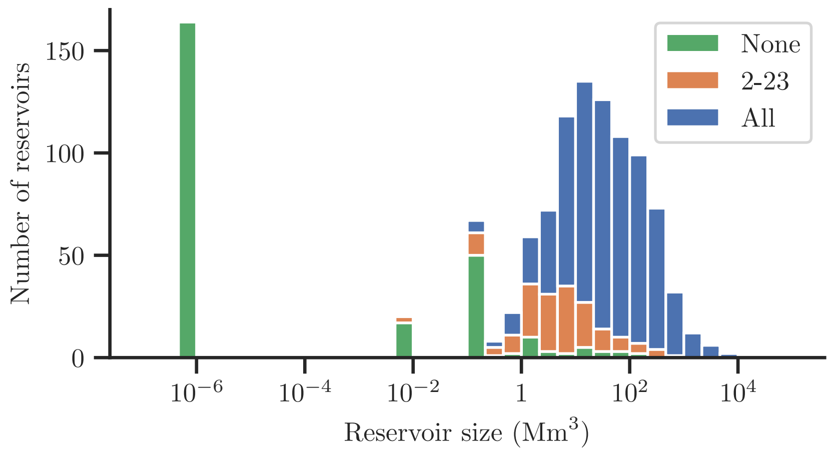


Figure 4.3: The figure shows how many of the water balances during a day were combined with respect to the reservoir size. The green bars represent reservoirs where none of the waterbalances were combined, the orange bars shows reservoirs with 2–23 combined water balances, while reservoirs where all 24 water balances were combined into one constraint are represented by the blue bars. The bin sizes follow a logarithmic scale.

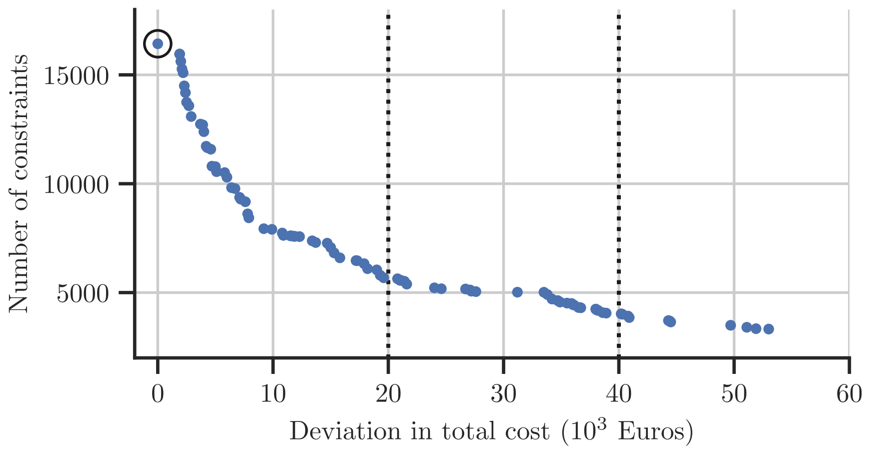


Figure 4.4: The figure shows the Pareto optimal values after running a genetic algorithm with the CES dataset for 30 generations. The base case, which serves as the reference for the total cost, has 16428 iterations and is indicated by a circle in the figure. Dotted lines mark thresholds for deviation in cost, and the constraints that are removed for the optimal individuals wihin those thresholds are represented in more detail in table 4.2.

Table 4.2: Table showing a detailed count of constraint types in the base case (Base), after removing constraints with a dual value equal to zero (DV), and using the optimal individual of the genetic algorithm with a threshold deviation of €20,000 (GA1) and €40,000 (GA2).

	Original	Base	GA1	GA2
Water Balances	26952	6199	2780	1283
Discharge Balances	26952	7905	2971	1739
Hydro Power Balances	768	768	768	768
Thermal Power Balances	192	192	192	192
Cable Ramping Upper Limit	3680	2	2	2
Cable Ramping Lower Limit	3680	2	2	2
Lower Reservoir Bounds	26952	17	17	17
Upper Reservoir Bounds	26952	1	1	1
Lower Bypass Bounds	26952	834	648	384
Upper Bypass Bounds	26952	1	1	1
Lower Discharge Limits	26952	104	104	104
Upper Discharge Limits	26952	400	80	65

of the number of iterations in the base case. The total number of constraints for this individual is 7566.

For comparison, the impact of choosing a threshold of €40,000 is also investigated. The number of iterations is reduced to 24% of the base case. The total number of constraints is now reduced to 4558, which is 2.0% of the original number of constraints.

From analyzing the corresponding Pareto optimal individual, it is found that the majority of the removed constraints, with respect to the base case, are water balances. Once again, the water balances for the smallest reservoirs are mostly kept, whereas the water balances for the bigger reservoirs can be removed. In fact, for several of the reservoirs, all water balances are removed altogether.

Cable ramping has little influence on the system in this particular case, and all constraints aside from two have a dual value of zero.

None of the power balances are removed. As previously seen, all the power balance restrictions remained after removing restrictions with a dual value equal to zero as well. Thus, all the power balances of the original problem are left intact after both procedures have been applied. This suggest that the power balances can be overlooked when using machine learning to search for constraints that can be removed.

4.5 Constraints Classification

4.5.1 Artificial Neural Network

Figure 4.5 shows that the loss curves are still descending in four out of the six models, and none of the validation curves have started to diverge from the training curves, which implies that none of the models suffers significantly from overfitting. Figure 4.6 shows the training accuracies and the Hamming losses of the models. Also here, the training and validation curves do not diverge in any of the plots. Except for the volume flow model, the training and validation curves eventually flattens. A possible cause is that the machine learning models do not have the capacity to capture all the details and patterns in the data set, even though the loss function is still decreasing.

It is noted that the validation losses are lower and that the validation accuracies are higher than the training losses and accuracies, respectively. During training, the model is penalized by the regularization, i.e. the dropout layers and L2-regularization, that is added to the model. During validation, the final model is applied to the data, without the regularization penalties. Thus, validation losses that are lower, and validation accuracies that are higher than their corresponding values for the training set, is to be expected.

Figure 4.6 also shows the differences between the different evaluation metrics. The binary accuracy and multi-label accuracy can give completely different pictures on the performance of the model.

Figure 4.8 shows graphs of the receiver operating characteristics for the final models when applied to the test set. The figure shows both the micro and macro-averaged curves, that are obtained by averaging the true and false positive rates of each label with either equation (2.5) or (2.6) in section 2.3.2. The area under the curves (AUCs) are indicated in the plot legends.

In all six models, both the macro-averaged and micro-averaged curves show that the models are able to capture information in the input data that enables the models to make more informed predictions than pure guesswork. At the same time, the ROC curves also show that there is room for improving the models. The macro average is more impacted by having few labels that fails to be properly classified, because every label is given equal weight, regardless of how often they occur [25]. The micro average, on the other hand, is more influenced by the performance of the most common labels, because the performance of each sample is given equal weight. The micro-averaged ROC curves for all six models in figure 4.8 tells that the model is able to predict the most common labels quite well, with micro-averaged AUC scores ranging from 0.86 to 0.94. The macro average AUC scores are likely to be reduced due to a minority subset of labels that the models fail to predict well. Infrequent labels are harder to predict with a machine learning model because of the smaller number of training samples. Therefore, lower macro-averaged AUC

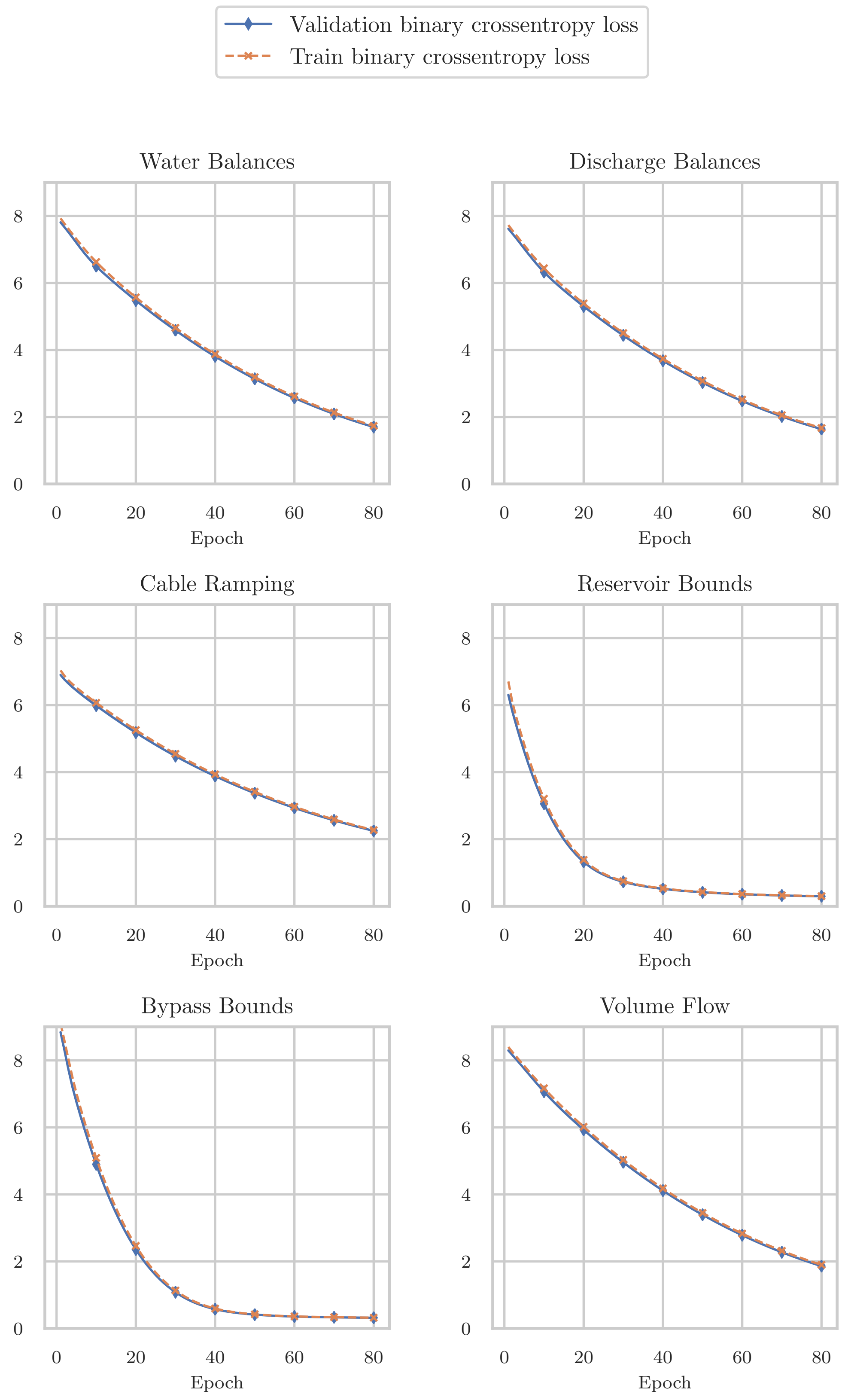


Figure 4.5: Training losses.

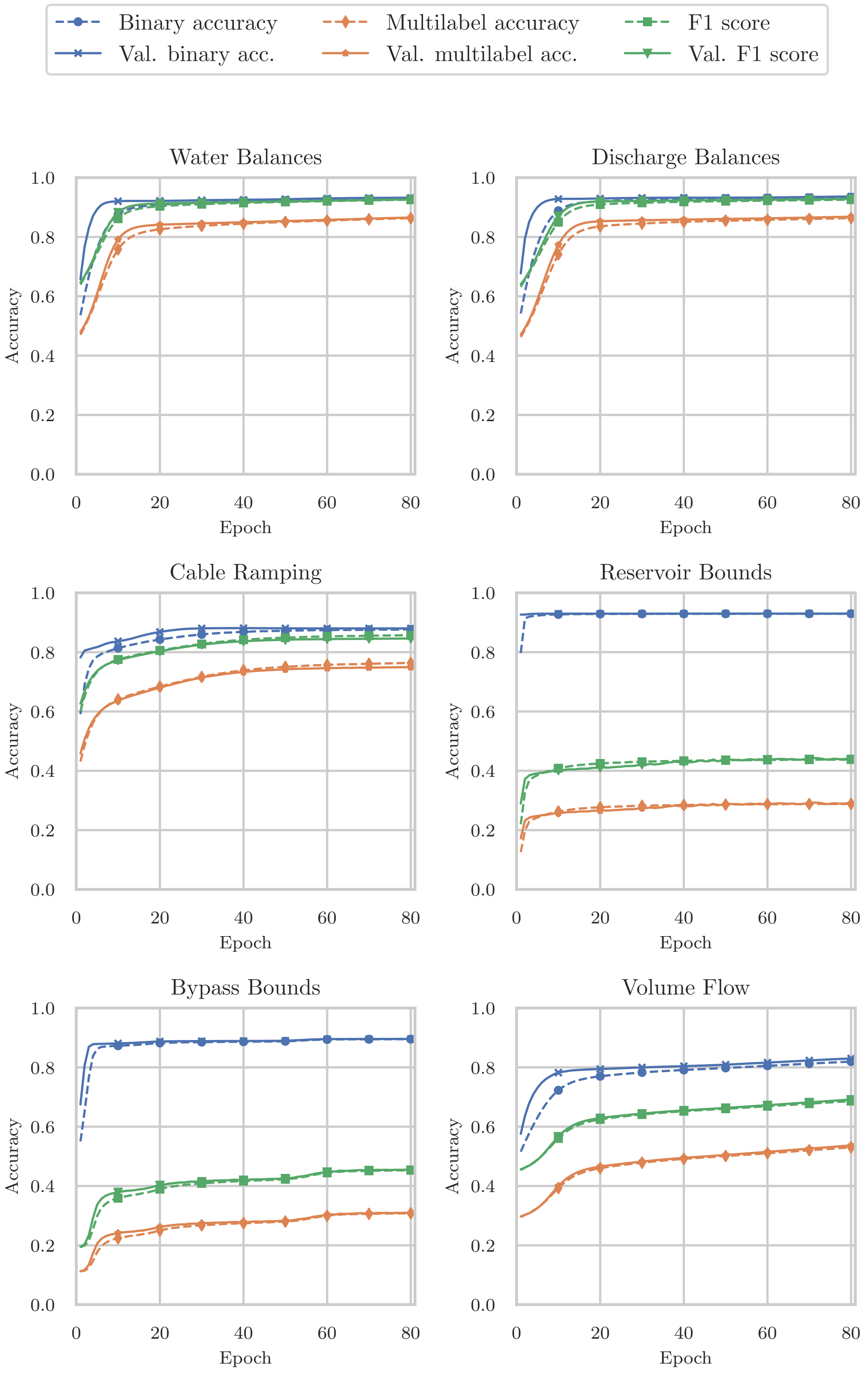


Figure 4.6: Training accuracies.

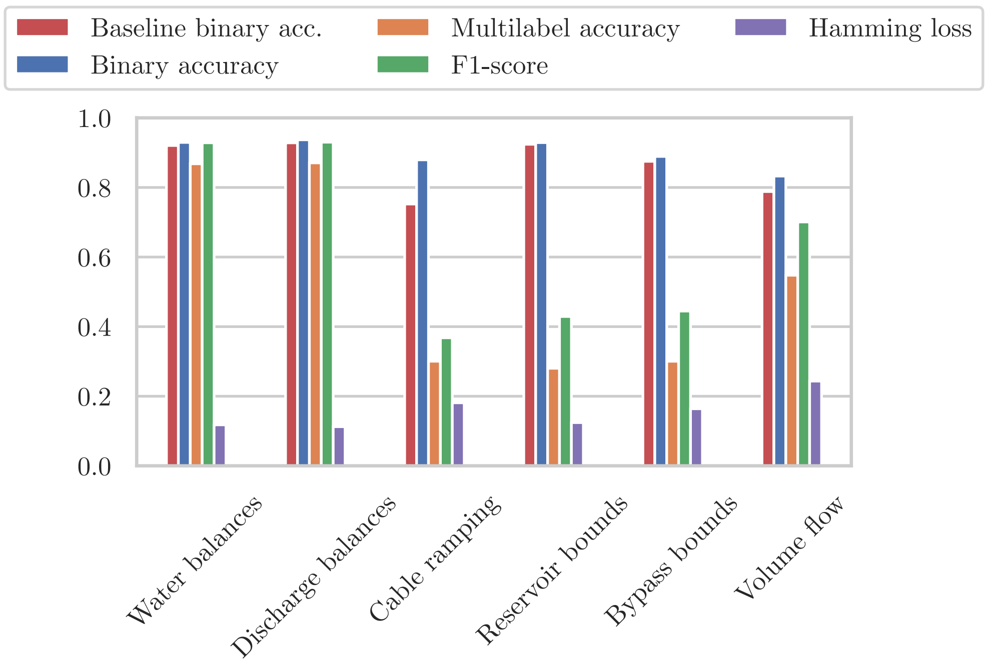


Figure 4.7: Accuracies achieved on the test partition of the data set.

scores are to be expected. A large gap between the two scores is symptomatic of an imbalanced label distribution.

By increasing the discrimination threshold when assigning class labels, i.e. increasing the required probability that a constraint can in fact be removed, the false positive ratio will increase, implying that fewer constraints are removed in total. At the same time, having a high true positive ratio, implies having fewer false negatives, because the total number of positive samples equals the sum of correctly predicted positive labels and incorrectly predicted negative labels (true positives + false negatives = positive labels). A low false negative rate should be prioritized to avoid that the error to the solution of the hydrothermal scheduling model becomes large. Thus, a true positive rate close to 1.0 is desired, while at the same time having a false positive rate as small as possible in order to remove as many constraints as possible. By studying the ROC curves in figure 4.8, the most promising classifiers are those for bypass bounds and volume flow restrictions. In these models, adjusting the discrimination threshold such that the false positive rate is around 0.4, the macro-averaged true positive rate will remain at a level above 0.9.

multi-label

4.5.2 Impact on Computational Times and Total Costs

In figure 4.9, the total system cost of the problems in the test set is seen, and the base cost computed using the unmodified problem is compared with the results from using the predicted

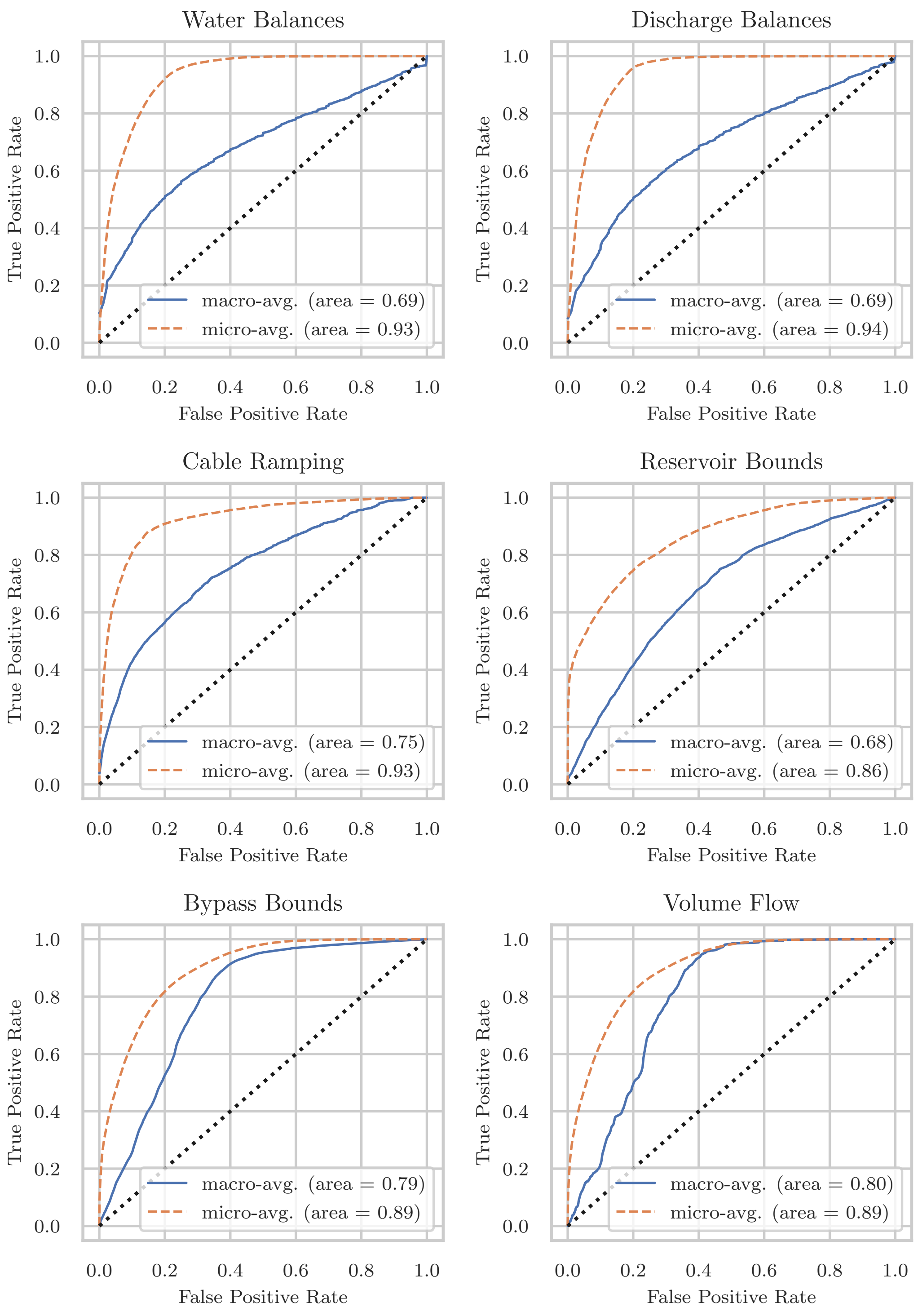


Figure 4.8: Receiver operating characteristic (ROC) curves related to the test performance of the machine learning models. The solid curves are the micro averages, and the dashed curves are the macro averages. The area under the curves for each of the curves (AUC) are indicated in the legend. The diagonal straight dotted line indicates the expected performance over time by randomly guessing each label.

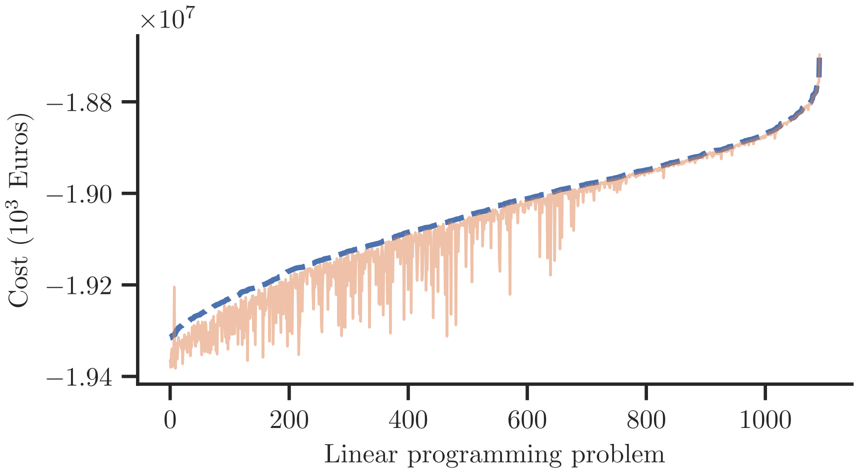


Figure 4.9: The dashed line shows the costs of the original problem, while the solid line shows the cost after deactivating reservoir balances and cable restrictions as suggested by the machine learning model. The test set consist of randomly selected daily problems, and the values are ordered by the calculated costs in the original problems.

relevant constraints. The mean absolute error (MAE) in cost is 2.73×10^4, which constitute a mean percentage error of 0.14%.

With lower costs, and thus higher profits, the modified model underestimate the system costs, indicating that the false positive ratio is too high and that too many restrictions have been removed. When the system costs are higher than -1.90×10^7, the system costs of the reduced models deviates little from the system costs of the original problems.

Figure 4.10 shows the achieved reduction in computation times with the data from the test split. The average computation time in deterministic ticks was reduced by 55%. Although the computation times are reduced, the average number of iterations increases by 12%.

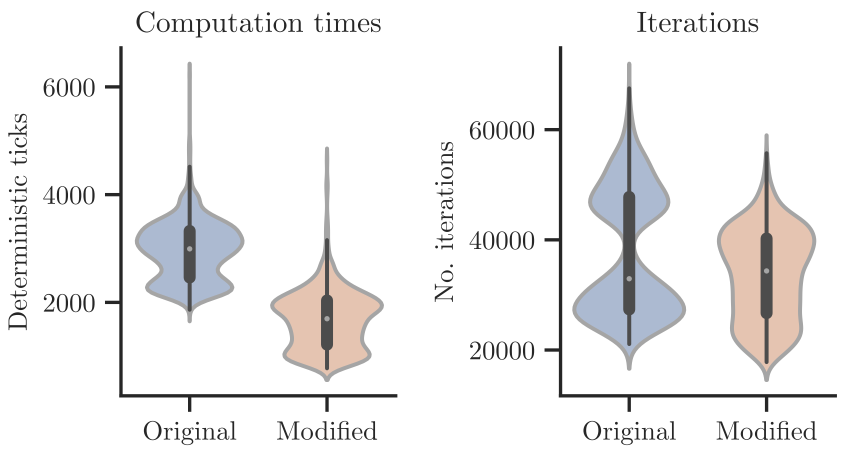


Figure 4.10: Violin plots showing the distribution of computation times (left) and the number of iterations (right) for the original and the reduced problems. The black boxes inside the plots describe the interquartile range, and the small white dots indicate the mean values.

Chapter 5

Discussion

5.1 Model Reduction Potential

Combining water balances of large reservoirs in consecutive time steps is a simple, yet effective method for reducing computation times. The analysis shows that the method can be used without affecting the result of the optimal solution. This method can easily be implemented in existing scheduling models, and can be used independently of any of the machine learning efforts that are presented in this thesis.

Smaller reservoirs have smaller water storage capacity, and are more prone to variations in inflow and release. The release and reduction in water volume is dependent on electricity supply and demand. These parameters are expected to be have a larger degree of variation and uncertainty in the future. Having small reservoirs as reserve capacity necessitates detailed modelling and a high temporal resolution as large spikes in demand or production can happen during small time intervals. A coarser resolution could result in the model not accurately predicting the available reserve capacity. The results presented from finding which constraints could be removed without impacting the total system costs confirms previous results on the importance of properly modelling smaller reservoirs in the system.

In the future electricity grid, other small forms of energy storage solutions will start to emerge, and future fundamental market models may model these in a similar way as the small reservoirs. The importance of modelling small water reservoirs imply the importance of modelling small scale, short term energy storage solutions as well.

As shown in figure 4.4, it is possible to reduce the number of constraints by 75% while having an insignificant impact on the result of the modelling. This result shows that there is a further potential for finding ways to reduce linear programming problems produced by the scheduling model. One way of achieving this is through the methodology established in section 3.7.

5.2 Machine Learning

Some constraints were always relevant, while others were never active. The constraints that never contributed to the price formation in the studied case, may become relevant with variation in the expected future cost, which is provided by the long term strategic model, exogenous market prices and reservoir levels. Similarly, constraints that never were identified as possible to remove may be possible to remove when changing the abovementioned parameters. Nevertheless, if training data does not contain any variation for some particular labels, it is not necessary to use these data for training the model. It can be useful to identify the constraints that never are significant and manually evaluate whether these constraints need to be included in the model.

There were no signs of overfitting during training, which implies that the neural networks can likely be improved by increasing the number of parameters, or increase the number of epochs during training. An increased number of parameters will result from adding more hidden layers and increasing the number of activation units in each layer. The accuracies achieved with the model on the test data, shown in figure 4.7 closely match the accuracies achieved during training (figure 4.6), with one notable exception: The accuracy of the model for cable ramping restrictions is much worse during testing than during training. The poor performance of this model compared with some of the other models is somewhat surprising given that this model had the least amount of labels. However, the mean imbalance label ratio is lower in the models that performed better during testing, and the poor result can come from an unlucky train-test split, which could be avoided by stratifying the data.

More varied and realistic input data is needed during the training to create a more generalizable model. Seasonal changes in water levels were not modelled and the problems do not represent realistic yearly variations; Only variations in inflow, wind power generation and load for one particular week of the year. The water levels only changed inside the week, and was reset at the start of each week. The presented framework does not limit the number of input types, and can be used with complete system simulations.

Adding reservoir levels as features is likely to improve the results, since this is information that is sent to the hydrothermal scheduling model from the previous time step, except at the start of the week, and was not accounted for in the machine learning model. Without this information, the artificial neural network had no means to take the changing reservoir levels into account. During the modelling of a machine learning model, it is hypothesized that the input data yields all information needed to give predictions. Without the reservoir levels, this is probably not true for the models that were created, because no variation in the time dimension was considered.

On a larger scale, the machine learning model should be able to identify the rare cases where the constraints that usually can be removed starts to impact the total cost. The presented artificial neural network is not able to do so because of the insufficient amount of training data for the labels occurring in fewer than one percent of the samples. Even though these labels rarely were active, they were always included when solving the reduced problems to avoid that they detriment the solution of the scheduling model.

A few different approaches for dealing with imbalance in multi-label classification problems have been proposed and should be investigated if improved machine learning models are to be created. One type of methods are sampling methods, that can be used to either generate new samples that correspond to minority classes, or delete samples of majority classes. The authors of [7] have developed an oversampling technique called MLSMOTE (multi-label Synthetic Minority Over-sampling Technique) with multi-label datasets in mind. MLSMOTE generates new unique synthetic samples with labels that occur seldom. Another method named REMEDIAL (REsampling multi-label datasets by Decoupling highly ImbAlanced Labels) can be used to decouple samples with high number of majority and minority labels [5]. The samples are split into two new samples where one contains the majority labels, and the other contains the minority labels. The number of labels in some of the models in the present work supersedes the number of labels in the datasets that the abovementioned methods were tested on, however. None of the datasets had more than 1000 labels, whereas more than 6000 labels were used to predict volume flow constraints. Therefore, the computational performance of these approaches need to be assessed. The authors of [40], points out the impracticality of these methods with many labels and instead propose a representation based sampling method.

Another way of dealing with the imbalance problem is to create or adapt machine learning algorithms that take the imbalance of classes into consideration. In [10], the authors adapt a structured forest algorithm to address imbalanced data. The authors note that the algorithm should be able to scale to a higher number of labels, but at the cost of prediction accuracy.

Much of the reduction in computational times as seen in the case where machine learning was used to remove constraints of both water balances and cable ramping restrictions was not due to the machine learning model itself, but because of knowledge about constraints that was believed to never have an impact on the system costs, based on the training data. These constraints were removed manually. For the water balances, however, few constraints were never relevant. Out of 26952 water balances, only 42 were never active. Only 9.2% of the water balances could be removed some of the time, to various degree, and out of these, only 29% of these were used in the machine learning model due to lack of training data.

The output values generated by the final sigmoidal layer are probabilities. Each label in a sample is given a class by rounding the probability value to zero or one. The precision of the hydrothermal model can be given more weight by increasing the discrimination threshold for removing constraints. This can be achieved by requiring that the estimated probability that a constraint is not relevant must be higher than 50% before it can be removed, and thereby reducing the false negative rate.

If the number of labels are increased, caused by adding more constraints and having a higher temporal resolution in the scheduling model, the number of parameters in the neural network will increase quadratically. To produce the results given here, the label space in the machine learning task was divided into partitions that were associated with different types of constraints. This was necessary with the number of training samples at hand. With an even higher number of constraints, the label spaces can be divded into even smaller partitions, e.g. by considering each modelled price area separately. Solutions that scale better to a larger number of labels may be needed as well, if the number of labels in each partition is increased.

An advantage of working with machine learning in connection with LP models, is that it is possible to generate an infinite amount of training data by adjusting the model input. The disadvantage of working with generated data rather than with empirical data, is the computational time and resources needed for generating the data. In working with empirical data, time and resources is needed for collecting data. When working with mathematical models, a working model must already be present.

Chapter 6

Conclusion

A large portion of the constraints in the hydrothermal scheduling model PriMod could be removed. When removing constraints with a dual value of zero in the optimal solution of the LP problems produced by PriMod, the problem size is reduced significantly. It was shown that the LP problems can be further reduced by the use of genetic algorithms. Only one problem was evaluated with the use of a genetic algorithm due to the amount of computing resources needed for evaluating the performance of each individual. Nevertheless, the potential to reduce the LP problems such that the number of iterations with the dual simplex algorithms is lowered, while having an insignificant impact on the optimal solution, was established.

Combining water balances in consecutive time steps in the daily problems is an effective approach for reducing the number of water balances. The algorithm can easily be implemented in the current hydrothermal scheduling model. It is somewhat challenging, however, to combine this work into the framework presented in 3.7.

The neural network presented in section 3.6.3 was used to attempt to classify whether constraints can be removed. In a satisfactory result, the false negative rate should be small, but the model should be able to remove enough constraints such that the decrease in computation times in the hydrothermal scheduling model are notable. Findings show that the models are able to capture some patterns in the data for making classification, but more complex models and more training data are needed to achieve prediction accuracies that are satisfactory. This is not surprising, given the large number of classes, and the fact that the task is a multi-label classification problem. Nevertheless, some progress has been made, and no barriers for markedly improving the models was found, given time and computational resources.

6.1 Further work

Ultimately, the presented framework for reducing computational times must be used with larger problems, where the need for reductions is greater. Improvements must be made to the machine learning models so that they can handle larger and more varied cases, and at the same time increase the prediction accuracies.

Several possible ways to improve the models have been suggested. First, the generation of more training data is needed. Second, more complex models that are able to capture more complex patterns are needed. Third, the class imbalance problem needs to be addressed.

The distribution of classes in the train, test and different validation folds was tackled using iterative stratification. The performance of this method became problematic when used on the entire data set with more than 9000 labels and 10,000 samples. This is another issue that need to be addressed.

Other machine learning models specifically adapted for multi-label learning, such as MLTSVM and MLkNN, which were briefly mentioned in section 2.3.1, can be attempted to solve the task at hand. These models can further be strengthened by the use of label embedding methods, such as e.g. LNEMLC presented in [52]. The authors of [52] noted that LNEMLC scales well to extreme multi-label scenarios, making it possible to employ adapted multi-label machine learning algorithms with large number of labels (e.g. more than 1000 labels).

Multiprocessing and clusters can be facilitated to overcome much of the computational challenges that were met during the work of this thesis. If the genetic algorithm is to be run with several different LP problems, the use of multiprocessing or clusters, or both, is necessary.

Appendix A

Bibliography

[1] M. Ardelean and P. Minnebo. HDVC Power Cables in the World. techreport EUR 27527, Joint Research Centre, 2015.

[2] T. Back, D. B. Fogel, and Z. Michalewicz, editors. *Basic Algorithms and Operators*. IOP Publishing Ltd., Bristol, UK, UK, 1st edition, 1999.

[3] K. P. Bennett and E. Parrado-Hernández. The interplay of optimization and machine learning research. *Journal of Machine Learning Research*, 7(Jul):1265–1281, 2006.

[4] W. Bi and J. Kwok. Efficient multi-label classification with many labels. In *International Conference on Machine Learning*, pages 405–413, 2013.

[5] F. Charte, A. Rivera, M. J. del Jesus, and F. Herrera. Resampling multilabel datasets by decoupling highly imbalanced labels. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 489–501. Springer, 2015.

[6] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera. Addressing imbalance in multilabel classification: Measures and random resampling algorithms. *Neurocomputing*, 163:3–16, 2015.

[7] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera. Mlsmote: Approaching imbalanced multilabel learning through synthetic instance generation. *Knowledge-Based Systems*, 89:385–397, 2015.

[8] W.-J. Chen, Y.-H. Shao, C.-N. Li, and N.-Y. Deng. Mltsvm: a novel twin support vector machine to multi-label learning. *Pattern Recognition*, 52:61–74, 2016.

[9] F. Chollet et al. Keras. `https://keras.io`, 2015.

[10] Z. A. Daniels and D. N. Metaxas. Addressing imbalance in multi-label classification using structured hellinger forests. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[11] J. Deane, G. Drayton, and B. . Gallachóir. The impact of sub-hourly modelling in power systems with significant levels of renewable generation. *Applied Energy*, 113:152–158, 2014.

[12] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature PPSN VI*, pages 849–858, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[13] K. B. Debnath and M. Mourshed. Forecasting methods in energy planning models. *Renewable and Sustainable Energy Reviews*, 88:297 – 325, 2018.

[14] Directorate-General for Energy. EU Energy in Figures. Technical report, European Commission, Sept. 2018.

[15] Directorate-General for Energy, Directorate-General for Climate Action, and Directorate-General for Mobility and Transport. EU Reference Scenario 2016. Energy, transport and GHG emissions. Trends to 2050. Technical report, European Commission, July 2016.

[16] ENTSO-E. 2018 Ten Year Network Development Plan. Technical report, European Network of Transmission System Operators for Electricity, 2018.

[17] European Commission. *Energy roadmap 2050*. Publications Office of the European Union, 2012.

[18] European Commission. Achieving the 10Europe's electricity grid fit for 2020. Communication COM(2015) 82 final, European Commission, 2015.

[19] European Council. Conclusions on 2030 Climate and Energy Policy Framework, 2014. EUCO 169/14.

[20] I. A. Farhat and M. E. El-Hawary. Optimization methods applied for solving the short-term hydrothermal coordination problem. *Electric Power Systems Research*, 79(9):1308–1320, 2009.

[21] N. Flatabø, A. Haugstad, B. Mo, and O. Fosso. Short-term and Medium-term Generation Scheduling in the Norwegian Hydro System under a Competitive Power Market Structure. In *EPSOM '98 Proceedings, ETH Zürich, Switzerland*, volume 1, Sept. 1998.

[22] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, July 2012.

[23] O. B. Fosso, A. Gjelsvik, A. Haugstad, B. Mo, and I. Wangensteen. Generation scheduling in a deregulated system. The Norwegian case. *IEEE Transactions on Power Systems*, 14(1):75–81, Feb. 1999.

[24] K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183 – 192, 1989.

[25] E. Gibaja and S. Ventura. A Tutorial on Multi-Label Learning. *ACM Computing Surveys*, 47, Apr. 2015.

[26] S. S. Girija. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *Software available from tensorflow.org*, 2016.

[27] I. Graabak, S. Jaehnert, M. Korpås, and B. Mo. Norway as a Battery for the Future European Power System—Impacts on the Hydropower System. *Energies*, 10(12), 2017.

[28] I. Graabak, M. Korpås, S. Jaehnert, and M. Belsnes. Balancing future variable wind and solar power production in central-west europe with norwegian hydropower. *Energy*, 168, 11 2018.

[29] M. Haugen et al. On the Importance of Detailed Thermal Modeling for Price Forecasting in Hydro-Thermal Power Systems. In preparation.

[30] A. Haugstad and O. Rismark. Price Forecasting in an Open Electricity Market based on System Simulation. In *EPSOM '98 Proceedings, ETH Zürich, Switzerland*, 1998.

[31] A. Helseth. PRIBAS – Pricing Balancing Services in the Future Nordic Power Market. Internet, Apr. 2017.

[32] A. Helseth, M. Haugen, S. Jaehnert, B. Mo, H. Farahmand, and C. Naversen. Multi-Market Price Forecasting in Hydro-Thermal Power Systems. In *15th International Conference on the European Energy Market (EEM)*, pages 1–5, June 2018.

[33] A. Helseth, B. Mo, A. L. Henden, and G. Warland. Detailed long-term hydro-thermal scheduling for expansion planning in the Nordic power system. *IET Generation, Transmission Distribution*, 12(2):441–447, 2018.

[34] A. Helseth, G. Warland, B. Mo, and O. B. Fosso. Samnett - The EMPS Model with Power Flow Constraints. Technical Report TR A7164, SINTEF, 2011.

[35] K.-H. Huang and H.-T. Lin. Cost-sensitive label embedding for multi-label classification. *Machine Learning*, 106(9-10):1725–1746, 2017.

[36] IBM Corporation. *IBM ILOG CPLEX Optimization Studio, CPLEX User's Manual*, 2015. Version 12 Release 6.

[37] L. Jiang and G. Hu. Day-ahead price forecasting for electricity market using long-short term memory recurrent neural network. In *15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 949–954. IEEE, 2018.

[38] A. Konak, D. W. Coit, and A. E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007, 2006. Special Issue - Genetic Algorithms and Reliability.

[39] K. Lachhwani. Application of neural network models for mathematical programming problems: A state of art review. *Archives of Computational Methods in Engineering*, pages 1–12, 2019.

[40] L. Li and H. Wang. Towards label imbalance in multi-label classification with many labels. *CoRR*, abs/1604.01304, 2016.

[41] J. Liu, W.-C. Chang, Y. Wu, and Y. Yang. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124. ACM, 2017.

[42] D. G. Luenberger and Y. Ye. *Linear and Nonlinear Programming*, volume 228 of *International Series in Operations Research & Management Science.* Springer International Publishing AG, fourth edition, 2016.

[43] J. Nam, J. Kim, E. L. Mencía, I. Gurevych, and J. Fürnkranz. Large-scale multi-label text classification—revisiting neural networks. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 437–452. Springer, 2014.

[44] Nord Pool AS. Ramping. Internet.

[45] J. Nowotarski and R. Weron. Recent advances in electricity price forecasting: A review of probabilistic forecasting. *Renewable and Sustainable Energy Reviews*, 81:1548 – 1568, 2018.

[46] S. Raschka and V. Mirjalili. *Python Machine Learning.* Packt Publihing Ltd., second edition, 2017.

[47] D. E. Rumelhart, G. E. Hinton, and W. R. J. Learning intemal representations by error propagation. *Parallel Disfributed Processing: Explorations in the Microstructures of Cognition, D. E. Rumelhart*, pages 318–362, 1986.

[48] G. Savvidis and K. Hufendiek. Variable Time Resolution in LP Electricity Market and Investment Models. In *15th International Conference on the European Energy Market (EEM)*, pages 1–5, June 2018.

[49] K. Sechidis, G. Tsoumakas, and I. Vlahavas. On the stratification of multi-label data. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 145–158. Springer, 2011.

[50] Statnett SF. Fleksibilitet i det nordiske kraftmarkedet, 2018-2040. Technical report, Statnett SF, 2018. In Norwegian.

[51] B. Szalkai and V. Grolmusz. Near perfect protein multi-label classification with deep neural networks. *Methods*, 132:50–56, 2018.

[52] P. Szymański, T. Kajdanowicz, and N. Chawla. Lnemlc: Label network embeddings for multi-label classifiation. *arXiv preprint arXiv:1812.02956*, 2018.

[53] P. Szymański, T. Kajdanowicz, et al. scikit-multilearn: A Python library for Multi-Label Classification. *Journal of Machine Learning Research*, 20(6):1–22, 2019.

[54] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[55] G. Tsoumakas, I. Katakis, and I. Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD'08)*, volume 21, pages 53–59. sn, 2008.

[56] U. Ugurlu, I. Oksuz, and O. Tas. Electricity price forecasting using recurrent neural networks. *Energies*, 11(5):1255, 2018.

[57] G. Warland, A. L. Henden, and B. Mo. Use of parallel processing in applications for hydro power scheduling – current status and future challenges. *Energy Procedia*, 87:157 – 164, 2016. 5th International Workshop on Hydro Scheduling in Competitive Electricity Markets.

[58] G. Warland and B. Mo. Stochastic optimization model for detailed long-term hydro thermal scheduling using scenario-tree simulation. *Energy Procedia*, 87:165 – 172, 2016. 5th International Workshop on Hydro Scheduling in Competitive Electricity Markets.

[59] H.-F. Yu, P. Jain, P. Kar, and I. Dhillon. Large-scale multi-label learning with missing labels. In *International conference on machine learning*, pages 593–601, 2014.

[60] M.-L. Zhang and Z.-H. Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition*, 40(7):2038–2048, 2007.

[61] M.-L. Zhang and Z.-H. Zhou. A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering*, 26(8):1819–1837, 2014.

[62] W. Zhang, J. Yan, X. Wang, and H. Zha. Deep extreme multi-label learning. In *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*, pages 100–107. ACM, 2018.

[63] S. Zhu, X. Ji, W. Xu, and Y. Gong. Multi-labelled classification using maximum entropy method. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 274–281. ACM, 2005.

[64] F. Ziel and R. Steinert. Probabilistic mid-and long-term electricity price forecasting. *Renewable and Sustainable Energy Reviews*, 94:251–266, 2018.

Appendix B

Code

B.1 Genetic algorithm

```
# Fitness function where both objectives are to be minimized.
creator.create("Fitness", base.Fitness, weights=(-1.0, -1.0))

# One individual is an array of binaries.
creator.create("Individual",
               array.array,
               typecode='b',
               fitness=creator.Fitness)

# Container that will contain individual, population and
# function objects, etc.
toolbox = base.Toolbox()

# Attribute generator
toolbox.register("attr_bool", random.randint, 0, 1)

# Register one individual:
toolbox.register("individual",
                 tools.initRepeat,
                 creator.Individual,
                 toolbox.attr_bool,
                 NUM_CONSTR)

# Register a population of individuals:
toolbox.register("population",
                 tools.initRepeat,
                 list, toolbox.individual)


def evalCost(individual):
    """Evaluates the objective functions."""
    # lp.solve_subset solves the linear programming problem
    # using the constraints defined by individual
    cost, time = lp.solve_subset(individual)

    num_constr = sum(individual)

    # Infeasible problems are penalized. lp.solve_subset()
    # returns None if the problem was not solved.
    if not cost:
        delta_cost = 1e18
        num_constr = 1e18
    else:
        delta_cost = abs(cost - TRUE_COST)

    time = 0 if time is None else time
    print('{:.5e} {:.5g} {:.2f}'.format(delta_cost,
                                        num_constr, time))

    return delta_cost, num_constr

# Two-point crossover, mutation by flipping bits and NSGA-II
# selection algorithm:
toolbox.register("evaluate", evalCost)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.02)
toolbox.register("select", tools.selNSGA2)


def main(ngen=30, mu=150, lambda_=1050,
          cxpb=0.6, mutpb=0.3, rnd_seed=1):
    random.seed(rnd_seed)

    pop = population(n=mu)
    hof = tools.ParetoFront()
    stats = tools.Statistics(
        lambda ind: ind.fitness.values)
    stats.register("avg", numpy.mean, axis=0)
    stats.register("std", numpy.std, axis=0)
    stats.register("min", numpy.min, axis=0)
    stats.register("max", numpy.max, axis=0)

    pop, logbook = algorithms.eaMuCommaLambda(
        pop, toolbox, mu, lambda_, cxpb, mutpb, ngen,
        stats, halloffame=hof)

    return pop, logbook, stats, hof
```

B.2 Neural network

B.2.1 Evaluation metrics

Custom metrics for Keras were made to calculate the multilabel accuracy, F1-score and Hamming loss.

```
import keras.backend as K

def ml_acc(y_pred, y_true):
    """Calculates multilabel accuracy."""
    y_true = K.cast(y_true, K.floatx())
    y_pred = K.round(K.cast(y_pred, K.floatx()))

    intersection = K.sum(y_true * y_pred, axis=1)
    union = (K.sum(y_true, axis=1) + K.sum(y_pred, axis=1)
             - intersection)

    score = K.map_fn(lambda x: 1.0 if x[1] == 0 else x[0]/x[1],
                     [intersection, union],
                     dtype=K.floatx())

    return K.mean(score)

def f1_score(y_pred, y_true):
    """Calculates the F1-score."""
    y_true = K.cast(y_true, K.floatx())
    y_pred = K.round(K.cast(y_pred, K.floatx()))
    intersection = K.sum(y_true * y_pred, axis=1)

    def calc_score(x):
        if K.sum(x[1]) + K.sum(x[2]) == 0:
            return 1.0
        else:
            return 2*K.sum(x[0])/(K.sum(x[1]) + K.sum(x[2]))

    score = K.map_fn(calc_score(x),
                     [intersection, y_pred, y_true],
                     dtype=K.floatx())

    return K.mean(score)

def hamming_loss(y_pred, y_true):
    """Calculates the Hamming loss."""
    y_true = K.cast(y_true, K.floatx())
    y_pred = K.cast(y_pred, K.floatx())

    return K.mean((1 - y_pred)*y_true + y_pred*(1 - y_true),
                  axis=1)
```

B.2.2 Model definition

```
def build_model():
    input_1 = Input(
        shape=(1035,), dtype=float, name='inflow_and_wind')
    output_1 = Dense(
        2*1035, activation='relu',
        kernel_regularizer=regularizers.l2(0.001))(input_1)

    input_2 = Input(
        shape=(24, 18), dtype=float, name='load')
    x = TimeDistributed(Dense(
        2*18, activation='relu',
        kernel_regularizer=regularizers.l2(0.001)))(input_2)
    output_2 = Flatten()(x)

    x = layers.concatenate([output_1, output_2])
    x = Dropout(0.5)(x)
    x = Dense(
        2*(1035+432), activation='relu',
        input_shape=(2*(1035+432),),
        kernel_regularizer=regularizers.l2(0.001))(x)
    x = Dropout(0.5)(x)
    x = Dense(
        1035+432, activation='relu',
        kernel_regularizer=regularizers.l2(0.001))(x)
    x = Dropout(0.5)(x)
    output_main = Dense(
        527, activation='sigmoid',
        kernel_regularizer=regularizers.l2(0.001))(x)

    model_db = Model(inputs=[input_1, input_2], outputs=output_main)

    model.compile(
        optimizer=optimizers.RMSprop(
            lr=1e-5, rho=0.9,
            epsilon=None, decay=0.0),
        loss='binary_crossentropy',
        metrics=[
            'binary_accuracy', ml_acc,
            f1_score, hamming_loss])

    return model
```

B.2.3 K-fold cross validation

Iterative k-fold cross validation was perfomed using the Python module iterstrat, which is part of a BSD 3 licenced Python package named iterative-stratification, available on GitHub[1].

[1] https://github.com/trent-b/iterative-stratification

```
from iterstrat.ml_stratifiers import MultilabelStratifiedKFold

history = dict()
mskf = MultilabelStratifiedKFold(n_splits=4, random_state=0)

for fold, (train, val) in enumerate(
        mskf.split(np.zeros((Y_train.shape[0], )), Y_train)):

    model = build_model()

    # Standardizing feature matrix for inflow and wind power.
    X_iw_train_avg = X_iw_train[train].mean()
    X_iw_train_std = X_iw_train[train].std()
    X_iw_train = (X_iw_train[train] - X_iw_train_avg)/X_iw_train_std

    # Standardizing feature matrix for load. Each channel is treated
    # separately.
    X_ld_train_avg = X_train[train].mean(axis=(0, 2), keepdims=True)
    X_ld_train_std = X_train[train].std(axis=(0, 2), keepdims=True)
    X_ld_train = (X_train[train] - X_ld_train_avg)/X_ld_train_std

    # Standardizing validation set using the standard deviation and
    # mean of the training set:
    X_ld_val = (X_train[val] - X_ld_train_avg)/X_ld_train_std
    X_iw_val = (X_train[val] - X_iw_train_avg)/X_iw_train_std

    history[fold] = model.fit(
        x=[X_iw_train, X_db_ld_train],
        y=Y_train[train],
        validation_data=[[X_iw_val, X_ld_val], Y_train[val]],
        batch_size=32,
        epochs=100)
```