Delft University of Technology
Faculty of Electrical Engineering, Mathematics & Computer Science
Delft Institute of Applied Mathematics

Machine learning to support cutting plane selection in two-stage robust optimization problems using a column-and-constraint-generation algorithm

Bachelor thesis project report
to obtain the degree of

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

BOAZ WILLEM VAN DER VLUGT

Delft, Netherlands
July 2021

BSc Thesis - Applied Mathematics

Machine learning to support cutting plane selection in two-stage robust optimization problems using a column-and-constraint-generation algorithm

BOAZ WILLEM VAN DER VLUGT

Delft University of Technology

Supervisor

Dr. K.S. (Krzysztof) Postek

Thesis committee

Dr. J.L.A. (Johan) Dubbeldam

July, 2021 Delft

Abstract

The field of robust optimization deals with problems where uncertainty influences the optimal decision. Some of these problems can be formulated in a 'two-stage' formulation, such as the location transportation problem. To solve such a problem, a column-and-constraint-generation algorithm has been introduced in which constraints are iteratively added to mixed-integer program based on different uncertain scenarios. However, if these scenarios are randomly chosen, this problem can grow too large to efficiently solve for. For most problems, there is some minimal set of scenarios needed to find the optimal solution, and it is important to find the 'right' scenarios early.

In this study, we attempt to predict these scenarios for a location transportation using machine learning. Using customer demand data for different instances of the problem, we train a logistic regression classifier, a neural network and a random forest classifier to predict important scenarios for newly generated problems. We find that when applying these machine learning tools, we reach an average reduction of scenarios added to the problem ranging from 8% to 24%. Even though we do not spend much effort on training perfect models, we see that there is a strong indication that machine learning can be used to increase the efficiency of the algorithm.

Contents

1 | Introduction

We are often confronted with the problem of determining the 'best' decision or course of action to take in a certain situation. The field of mathematical optimization provides us with a broad set of tools to help us determine this desired optimal result, taking into account constraints of the problem and the various factors that may influence the final decision. The method of linear programming, for instance, is a very straightforward method of modelling a problem in which we want to find a maximum or minimum value of some objective function, subject to a set of constraints (given that the problem can be represented by linear relationships). Various algorithms have been developed to solve this type of problem, providing great advances in fields such as operations research and decision theory.

One drawback to simple methods like linear programming is that most real-life situations are more complex than what can be represented using such a model. One of the most important factors in this complexity is the presence of uncertainty. In most practical problems that we encounter in the field of optimization, the outcome of the problem is affected to some degree by factors of which we are not completely certain what their final value/outcome will be. In some cases we can make safe assumptions about the outcome of some variable, but most of the time we will deal with variables that could take on a wide range of values, and the actual values will only be known to us after we have made (part of) a decision.

To deal with the presence of uncertainty, there exists a specific field of optimization known as robust optimization. Robust optimization methods, as the name suggests, are focused around finding an optimal solution that will work even in the 'worst case' scenario that might occur; we seek a certain level of robustness against the uncertainty. However, in many problems this can lead to a solution that is overly conservative; the potential worst-case scenario could be very far from the actual realization of the uncertainty. To combat this, methods have been developed in a specific field of robust optimization known as 'multi-stage optimization', where certain parts of the decision-making process are left until after the realization of uncertainty has been revealed.

One type of algorithm that can be applied to two-stage optimization problems is the 'column-and-constraint generation' (C&CG) algorithm, in which we search for an optimal solution by gradually building up a set of potential scenarios for which we can find a working solution until we have a solution that is robust against all realizations of uncertainty. A weakness of this algorithm is that, when dealing with more elaborate problems, this set of scenarios can grow so large that the solution cannot be efficiently computed if the set is built up randomly. Thus, it can be beneficial to find a way of identifying the 'important' scenarios for a certain problem.

One way in which we might be able to identify these scenarios is using machine learning (ML), a field of study that encompasses the use of certain algorithms to make predictions based on past

data. A key feature of most machine learning algorithms is that they grow more accurate through experience; feeding more data into the algorithm improves the predictions that it makes. By training different ML models to identify important scenarios in existing two-stage optimization problems, we might be able to improve the efficiency of the C&CG algorithm.

To make the idea of two-stage optimization more concrete, we will introduce a real-life problem to which we can apply the C&CG procedure, namely the location transportation problem. In this problem, we examine a number of facilities and a number of customers in a certain area. We make two decisions: firstly, we must decide which facilities to open and the level of production at each facility. Then, once we have more information about the customer demand, we can decide how many products to transport from each facility to each customer to meet the demand, minimizing the fixed, variable and transport costs.

In this thesis, we will introduce the mathematical background behind robust and two-stage optimization problems (in Chapter 2). We will explain the C&CG algorithm in-depth, and show how it can be applied to the location transportation problem (introduced in Chapter 3). Then, in Chapter 4, we will examine several machine learning methods and apply them to the C&CG algorithm in a fashion as described above. Finally, we will present the results obtained in Chapter 5 and discuss the efficiency of the algorithm when supported by ML. The main research question that we aim to answer is: 'How can machine learning models be used to improve the efficiency of the C&CG algorithm?'.

2 | Mathematical Background

In this chapter, we introduce various concepts and terms related to robust and two-stage optimization problems, giving a thorough mathematical background for the problems discussed in this thesis. Furthermore, we give an explanation of the C&CG algorithm that can be applied to solve such problems.

2.1 Two-stage robust optimization

2.1.1 Linear programming

We first shortly mention the general formulation of a linear programming (LP) problem. For $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{n \times m}$, we formulate an LP problem in the following way:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \geq b \\ & x \geq 0 \end{aligned}$$

Here, $c^T x$ is the objective function (with $x \in \mathbb{R}^n$ the solution vector), i.e. the function to be minimized (or maximized). $Ax \geq b$ represents the constraints of the problem (with A and b given data about the problem), i.e. the restrictions that the final solution vector x must satisfy [1]. In general, solutions to LP problems can be efficiently computed using a solver.

For simplicity, we will denote the above formulation with the following (more compact) notation:

$$\min \left\{ c^T x | Ax \geq b, \quad x \geq 0 \right\}$$

2.1.2 Robust counterparts to LP problems

In the LP formulation presented in the previous section, we have that the matrix A and the vector b contain data about the circumstances of our problem. In most real world problems, this data is uncertain to some degree, meaning we do not know what the exact value of this data will be. There are several different approaches to dealing with this uncertainty. We could choose to ignore the uncertainty altogether and simply use the most likely or expected value. This simplifies the optimization problem, but it also means that we ignore a great deal of knowledge we have about our problem. Another approach is to use stochastic programming methods, in which we assume the uncertainty follows a probability distribution and take this into account in the optimization of our problem.

The approach we will focus on in this thesis lies in the field of robust optimization. Such methods aim to find a solution to the optimization problem such that the constraints of the problem are satisfied for any possible final realization of (A, b). To define the possible realizations of (A, b), we introduce the concept of an uncertainty set.

We denote the uncertainty set with $\mathcal{Z}$ and define it to be the set of all possible values that (A, b) have a positive probability of taking on. The size of $\mathcal{Z}$ differs per problem, and may be either a finite set of discrete values or a continuous range. Whether we are dealing with a finite discrete set plays an important role in the approach to solving the optimization problem; for finite uncertainty sets, each scenario simply adds a constraint to the problem. This is not possible when the uncertainty set is infinite.

We now introduce the robust counterpart of the uncertain LP problem from the previous section. We formulate a robust LP problem in the following way:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \geq b \quad \forall (A, b) \in \mathcal{Z} \\ & x \geq 0 \end{aligned}$$

Again, we can denote this using a more compact notation:

$$\min \left\{ c^T x | Ax \geq b \quad \forall (A, b) \in \mathcal{Z}, \quad x \geq 0 \right\}$$

The solution vector x^* to the above problem is referred to as a *robust solution* of the uncertain problem [2].

2.1.3 Two-stage optimization formulation

Clearly, finding a robust solution to an uncertain linear programming problem provides us with a rather 'safe' solution; it is likely that the realization of the uncertain data is not quite the worst case that we have optimized for. In some cases this cannot be avoided, as the smallest violation of the problem's constraints can have significant consequences. For instance, when designing large structures in engineering, we have a certain margin of security that we must adhere to when dealing with the various forces applied the material.

However, there are situations in which we can be more flexible when we make certain decisions, and we can delay some decisions until we know more about the uncertain data. In this case, it is useful to split the decision-making process into multiple parts. This is known as multi-stage optimization; in this thesis, we will focus specifically on dividing the decision in two stages, i.e. two-stage optimization. The key difference between the two stages is that the realization of uncertainty is revealed after the first-stage decision is made. Thus, we make a robust decision about some variables in the first stage, and once we have more exact knowledge about the circumstances of our problem, we can make our second-stage decision for the remaining variables (taking into account the fact that the first-stage decision cannot be modified anymore).

In general, we formulate such a two-stage optimization problem in the following manner:

$$\begin{aligned} \min_{x} \quad & c^T x + \max_{z \in \mathcal{Z}} \quad \min_{y \in F(x,z)} \quad b^T y \\ \text{s.t.} \quad & Ax \geq b \\ & x \geq 0, \quad y \geq 0 \\ \text{where} \quad & F(x,z) = \{y : Gy \geq h - Ex - Mz\} \end{aligned}$$

We see that in this formulation, we have two decision vectors; x is the first-stage decision, and y is the second-stage decision. The first component of the objective function is the same as in the previously presented formulations of (robust) LP problems. We see that in the second component, we minimize an objective function of the second-stage decision vector subject to constraints that take into account the first-stage decision as well as the possible realizations of uncertainty. This function is maximized over all realizations of uncertainty to take into account the worst-case uncertain scenario.

As previously mentioned, there is a significant difference in solving the multi-stage optimization problem depending on whether the uncertainty set is finite and discrete. If this is indeed the case, i.e. if $\mathcal{Z} = \{z_1, \ldots, z_n\}$, we define the corresponding recourse decision variables $\{y^1, \ldots, y^n\}$ to be the optimal decision given this realization of uncertainty. We can then reformulate the two-stage optimization problem as follows:

$$\begin{aligned} \min_{x} \quad & c^T x + \eta \\ \text{s.t.} \quad & Ax \geq b \\ & \eta \geq b^T y^i, \quad i = 1, \ldots, n \\ & Ex + Gy^i \geq h - Mz_i, \quad i = 1, \ldots, n \end{aligned}$$

The second constraint ensures that the objective function of the second-stage decision is minimized. Thus, when the uncertainty is a finite discrete set, the problem is simply a mixed integer program. Depending on the number of scenarios in the uncertainty set, this MIP can quickly grow to be quite a large-scale problem. It can clearly be seen that, for an uncertain two-stage minimization problem, solving the problem for a subset of the uncertainty set provides an objective value that will in any case not exceed the objective value of the original problem. This is due to the fact that removing scenarios (and thus constraints) makes the problem easier to solve, allowing for a lower minimum value, providing a lower bound to the true objective value [3].

2.2 Column & constraint generation algorithm

In the previous section, we discussed the procedure of finding a lower bound for the objective value of a minimization problem by solving the problem for a (finite discrete) subset of the uncertainty set. This method is an important component of the column-and-constraint generation (C&CG) algorithm [3] for solving two-stage optimization problems. Such an algorithm is especially useful in cases where the uncertainty set is not finite or discrete, or when the uncertainty set is very large.

The essential idea behind the C&CG algorithm is to start with a small discrete initial subset of the entire uncertainty set, and to find an optimal solution that satisfies the constraints corresponding

to these scenarios. This provides us with a lower bound for the true objective value. Using this decision, we then search for an upper bound to the objective value, or identify a new scenario for which our current decision is infeasible. We add this scenario to our problem and continue iterating until the difference between the upper and lower bound is sufficiently small.

The algorithm works with a master-subproblem framework to generate bounds and scenarios. The implementation of the algorithm and the exact formulation of these problems is given in the next sections.

2.2.1 Algorithm

The C&CG algorithm can be implemented in the following way: [3]

Algorithm 1: Column-and-Constraint Generation Algorithm

```
LB = −∞
UB = +∞
k = 0
𝒰 = ∅

Solve the MP to obtain an optimal solution (x*_{k+1}, η*_{k+1}, y^{1*}, ..., y^{k*})
LB = c^T x*_{k+1} + η*_{k+1}

Solve the SP to obtain an optimal solution y^{k+1}, optimal value 𝒬(x*_{k+1}) and scenario z*_{k+1}
UB = min{UB,  c^T x*_{k+1} + 𝒬(x*_{k+1})}

if UB − LB ≤ ε then
    Return x*_{k+1} and terminate
else
    if 𝒬(x*_{k+1}) < +∞ then
        Create variables y^{k+1}
        Add the following constraint to the MP: η ≥ b^T y^{k+1}
        Add the following constraint to the MP: Ex + Gy^{k+1} ≥ h − M z*_{k+1}

        k = k + 1
        𝒰 = 𝒰 ∪ {k + 1}
    else
        Create variables y^{k+1}
        Add the following constraint to the MP: Ex + Gy^{k+1} ≥ h − M z*_{k+1}

        k = k + 1
```

In the implementation of the algorithm above, 'MP' and 'SP' refer to the master problem and subproblem (which are explained in the next sections).

2.2.2 Master problem

The master problem (MP) of the C&CG algorithm is a mixed-integer program where we solve the two-stage optimization problem for an enumerated subset of the uncertainty set. Given a

finite discrete set $\mathcal{U} \subset \mathbb{N}$ and a positive integer $k \in \mathbb{N}$ such that $|\mathcal{U}| \leq k$, we formulate the following master problem:

$$\begin{aligned} \min_{x,\eta} \quad & c^T x + \eta \\ \text{s.t.} \quad & Ax \geq b \\ & \eta \geq b^T y^i, && \forall i \in \mathcal{U} \\ & Ex + Gy^i \geq h - Mz_i, && \forall i \leq k \end{aligned}$$

Here, z_i are enumerated scenarios from the uncertainty set that have been identified using the subproblem (which will be explained in the next section), and y^i are second-stage decisions corresponding to scenarios from the uncertainty set.

2.2.3 Subproblem

The subproblem (SP) of the C&CG algorithm is solved following the master problem in every iteration. The master problem gives us an optimal first-stage decision x^* for the currently examined scenarios from the uncertainty set. Given this first-stage decision, we formulate the following subproblem:

$$\begin{aligned} \mathcal{Q}(x^*) = \quad & \max_{z \in \mathcal{Z}} \quad \min_{y} \quad b^T y \\ \text{s.t.} \quad & Ex^* + Gy \geq h - Mz \end{aligned}$$

The subproblem searches for a scenario z^* in the full uncertainty set and a corresponding second-stage decision y^* that minimizes the second-stage cost of the problem $\mathcal{Q}(x^*)$ given the first-stage decision.

It is also possible that we find a scenario for which no second-stage decision is feasible given the first-stage decision; in this case, we set $\mathcal{Q}(x^*) = +\infty$.

It is important to note that we need some oracle to be able to solve this subproblem. This will be elaborated on at a later moment in this thesis, when we discuss the application of the C&CG algorithm to the location transportation problem.

3 | Location Transportation Problem

In order to make the idea behind two-stage optimization problems and the application of the C&CG algorithm more concrete, we introduce a real-life problem that can be solved using such an approach: the location transportation problem [3]. For the remainder of this thesis, we will refer to this problem when discussing the algorithm and results.

3.1 Variables of the problem

The location transportation problem deals with the optimization of a given distribution network in an area for a certain product. There are two main questions we attempt to answer:

1. Given S production facilities located at various different locations in the area, which of the facilities should be open or closed, and how much of the product should each facility produce?

2. Given C customers in the area, how many products should we transport from each facility $s = 1, \ldots, S$ to each customer $c = 1, \ldots, C$ in order to satisfy (uncertain) customer demand?

This already gives an indication of how we can formulate the problem as a two-stage optimization problem. The first question encompasses the first-stage decision to be made, and the second question is our second-stage decision that we make after the uncertainty (i.e. the customer demand) has been revealed. We can encode these decisions in the following decision variables:

- $\mathbf{x} \in \{0, 1\}^S$: Binary variable indicating whether facility s is open.
- $\mathbf{p} \in \mathbb{R}_+^S$: Continuous variable indicating production level at facility s.
- $\mathbf{y} \in \mathbb{R}_+^{S \times C}$: Continuous variable indicating number of products transported from facility s to customer c.

Next, to further formulate our optimization problem, we examine the objective function. In location transportation problems, we usually aim to minimize our total costs. Naturally, we deal with fixed costs and variable costs, which we can formulate in the following parameters:

- f_s: Fixed cost of opening facility s.
- v_s: Variable cost producing one unit of the product at facility s.
- $t_{s,c}$: Variable cost of transporting one unit of the product from facility s to customer c.

Finally, we introduce the uncertainty. As mentioned previously, customer demand is usually the uncertain factor in location transportation problems. Although we can make a reasonably accurate prediction regarding the demand distribution, we accept that the exact demand is only known after we make our first-stage decision. We assume some 'nominal' demand for customer c, denoted with $\overline{d}_c$, and take some value δ_c to be the maximum deviation from the nominal value. We then express customer demand as the following:

$$d_c = \overline{d}_c + z_c \delta_c,$$

where $\mathbf{z} = (z_1, \ldots, z_C) \in \mathcal{Z}$, with

$$\mathcal{Z} = \left\{ \mathbf{z} \in \mathbb{R}_+^C : \mathbf{0} \leq \mathbf{z} \leq \mathbf{1}, \sum_{c=1}^{C} z_c \leq \Gamma \right\}.$$

The uncertainty set $\mathcal{Z}$ has a certain 'budget', $\Gamma \leq C$, which indicates the proportion of values of $\mathbf{z}$ that are allowed to be maximum. If $\Gamma = C$, for instance, the uncertainty set will contain the scenario in which the customer demand deviates maximally from the nominal value for all customers.

3.2 Problem formulation

Now that we have defined the necessary variables, we can formulate the complete optimization problem with constraints. The problem is defined as follows:

$$\min_{\mathbf{x} \in \{0,1\}^S, \mathbf{p} \in \mathbb{R}_+^S} \quad \sum_{s=1}^{S} (x_s f_s + p_s v_s) + \max_{\mathbf{z} \in \mathcal{Z}} \quad \min_{\eta, \mathbf{y} \in F(\mathbf{x}, \mathbf{p}, \mathbf{z})} \quad \eta \tag{3.1}$$

$$\text{s.t.} \quad p_s \leq M x_s \qquad \forall s = 1, \ldots, S \tag{3.2}$$

$$\text{where} \quad F(\mathbf{x}, \mathbf{p}, \mathbf{z}) = \left\{ \eta \in \mathbb{R}, \quad \mathbf{y} \in \mathbb{R}_+^{S \times C} \right\} \text{ such that:}$$

$$\eta \geq \sum_{s=1}^{S} \sum_{c=1}^{C} y_{s,c} t_{s,c} \tag{3.3}$$

$$\sum_{c=1}^{C} y_{s,c} \leq p_s \qquad \forall s = 1, \ldots, S \tag{3.4}$$

$$\sum_{s=1}^{S} y_{s,c} \geq \overline{d}_c + z_c \delta_c \qquad \forall c = 1, \ldots, C, \quad \forall \mathbf{z} \in \mathcal{Z} \tag{3.5}$$

Here, (3.1) is the objective function of our problem; we seek the first-stage decision $(\mathbf{x}, \mathbf{p})$ that minimizes the fixed costs of opening facilities and the variable costs of production, and the second stage-decision $\mathbf{y}$ that minimizes η (which is equal to the second-stage cost due constraint (3.3)). We maximize this minimum second-stage cost over the uncertainty set to obtain the 'worst-case' total cost.

Constraint (3.2) ensures that if a facility is closed, the production level is zero for that facility. If the facility is open there is no restriction on the production level, indicated by the 'big M'.

In theory, we could let this M be ∞; however, computation-wise it is better to avoid M being too large. We know that no facility will ever need to produce more than the total sum of the maximum possible customer demand, so we let $M = \sum_{c=1}^{C} \overline{d}_c + 1 \cdot \delta_c$ for our problem, using the fact that $z_c \leq 1$.

Constraint (3.4) ensures that the total amount of product that we transport from any facility doesn't exceed the production level at the facility.

Finally, constraint (3.5) is the uncertainty constraint. We ensure that the total amount of product that we supply to a certain customer meets the customer demand for any scenario in the uncertainty set.

3.3 Application of the C&CG algorithm

3.3.1 Master problem

When the C&CG algorithm is applied to the location transportation problem, the master problem is formulated as follows:

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{p},\mathbf{y^1},\ldots,\mathbf{y^N}} \quad & \sum_{s=1}^{S}(x_s f_s + p_s v_s) + \eta & & \\
\text{s.t.} \quad & \eta \geq \sum_{s=1}^{S}\sum_{c=1}^{C} y_{s,c}^{i} t_{s,c} & & \forall i = 1,\ldots,N \\
& p_s \leq M x_s & & \forall s = 1,\ldots,S \\
& \sum_{c=1}^{C} y_{s,c}^{i} \leq p_s & \forall s = 1,\ldots,S, & \forall i = 1,\ldots,N \\
& \sum_{s=1}^{S} y_{s,c}^{i} \geq \overline{d}_c + z_c \delta_c & \forall c = 1,\ldots,C, & \forall i = 1,\ldots,N
\end{aligned}
$$

Here, we have a finite set of scenarios $\hat{\mathcal{Z}} \subseteq \mathcal{Z}$ as an input, with $\mathbf{z^1},\ldots,\mathbf{z^N} \in \mathcal{Z}$. For every scenario in the 'current' uncertainty set, we look for an optimal second-stage decision $\mathbf{y^i}$.

3.3.2 Subproblem

As mentioned previously, the subproblem of the C&CG algorithm requires some oracle to find a scenario in the uncertainty set for which the currently found solutions are infeasible. For the location transportation problem, we can formulate the optimization problem for this oracle as:

$$
\begin{aligned}
\max_{\mathbf{z},\mathbf{b},\zeta} \quad & \zeta & & \\
\text{s.t.} \quad & \mathbf{z} \in \mathcal{Z}, \quad \mathbf{b} \in \{0,1\}^{N \times C}, \quad \zeta \in \mathbb{R} & & \\
& \sum_{c=1}^{C} b_c^i = 1 & & \forall i = 1,\ldots,N \\
& \zeta + b_c M_s \leq -\sum_{s=1}^{S} y_{s,c}^{i} + (\overline{d}_c + z_c \delta_c) + M & \forall c = 1,\ldots,C, & \forall i = 1,\ldots,N
\end{aligned}
$$

Again, M_s is a 'big M' just as in the problem constraints. For computations, we can take the value of this M to be the same as before.

The variable ζ indicates how much the demand constraint is violated for the found scenario $\mathbf{z}$. If $\zeta > 0$, we know that for all currently found second-stage decisions $\mathbf{y_1}, \ldots, \mathbf{y_N}$ at least one demand constraint is violated. In this case, we add the found scenario to the master problem in order to find a new decision that works for this scenario. If $\zeta < 0$, our current solution is feasible for all scenarios in the uncertainty set so it is robust, meaning we can terminate the algorithm.

3.3.3 Identification of important scenarios

As mentioned previously, every iteration of the C&CG algorithm adds a new violating scenario to the problem until we have a set of scenarios that 'covers' the entire uncertainty set $\mathcal{Z}$ (i.e. our first-stage decision $(\mathbf{x}, \mathbf{p})$ and at least one second-stage decision $\mathbf{y^i}$ are feasible for all realizations $\mathbf{z} \in \mathcal{Z}$). For every problem, there is a certain smallest set of scenarios such that the entire uncertainty set is covered; however, the C&CG algorithm doesn't necessarily look for these 'most important' scenarios which may cause the set (and subsequently the master problem) to grow too large to efficiently solve for.

In this thesis, we aim to improve the C&CG algorithm by finding a way to identify/predict these important scenarios. For this, it is important to mention a few properties of the uncertainty set.

Firstly, we have that if the uncertainty set is a polytope (which is the case for our problem) and $\text{ext}(\mathcal{Z})$ denotes the set of extreme points of the polytope, then:

$$\sum_{s=1}^{S} y_{s,c}^i \geq \overline{d}_c + z_c \delta_c \quad \forall \mathbf{z} \in \mathcal{Z}$$

$$\Longleftrightarrow$$

$$\sum_{s=1}^{S} y_{s,c}^i \geq \overline{d}_c + z_c \delta_c \quad \forall \mathbf{z} \in \text{ext}(\mathcal{Z})$$

This already reduces our uncertainty set to a finite set. However, the set may still be very large which can make computations difficult. We note another useful property. In our problem we are dealing with a budgeted uncertainty set. In finding an optimal decision for our problem, we only need to take into account scenarios for which the maximum budget Γ is used. Thus, we can reduce our uncertainty set to:

$$\dot{\mathcal{Z}} = \left\{ \mathbf{z} \in \text{ext}(\mathcal{Z}) : \sum_{c=1}^{C} z_c = \Gamma \right\}$$

When implementing the C&CG, we will make use of a Python package named *Pypoman* [4] to find $\dot{\mathcal{Z}}$. Using a halfspace representation of the entire uncertainty polytope, the package calculates the vertices for us. However, for high-dimensional uncertainty sets (depending on the number of customers C), computation times can become very high. This is an important factor in our choice for C in our implementation, which we discuss in the next section.

4 | Machine Learning and the C&CG Algorithm

The field of machine learning focuses on the development of techniques to identify potential underlying patterns in a set of data, and to make predictions using these patterns based on new data. This can be achieved using some existing model, which in essence is a (usually mathematical) process for generating output data based on a given input, using variable parameters. Such a model must be 'trained', which means that we (automatically) optimize the parameters of the model such that the output of the model is as close as we desire (based on a performance indication function of our own choosing) to some target value.

In this thesis, we study the application of machine learning techniques to the C&CG algorithm with the aim of predicting the 'important' scenarios for location transportation problems as introduced in Section 3.3.2. More specifically, we aim to predict solutions to the subproblem such that we need to solve as few subproblems as possible in the end. This section will provide some background information on the techniques applied, as well as an explanation on how exactly we alter the C&CG to incorporate ML-predictions into the algorithm.

4.1 Machine learning techniques

We will briefly provide some context regarding a few different machine learning techniques in order to be able to discuss the results obtained from applying them to the C&CG. Note that in this thesis, we are not aiming to train/tune a model to make perfect predictions; rather, we investigate if there is some indication that important scenarios can be predicted by training a ML-tool on existing data from location transportation problems.

Linear regression

A regression problem focuses on mapping some input vector $\mathbf{x}$ to a real-valued output $f(\mathbf{x})$. A set of training inputs $\mathbf{x_n}$ is given with corresponding outputs $y_n = f(\mathbf{x}) + \epsilon$, where ϵ describes 'noise' in the output values. Various models can be assumed for this noise. The function f must be inferred, usually according to some given function class (e.g. linear functions, polynomials, etc.) with parameters that can be varied to find a best fit for the training data.

In linear regression (as the name suggests), the function f is assumed to be linear in the parameters $\mathbf{a} = (a_1, \ldots, a_k)$. This means that the function is a linear combination of the 'input features'. A feature in this case refers to a function of the input, $\phi(\mathbf{x})$. Thus, we can for example

assume that f has the following form:

$$\mathbf{y} = f(\mathbf{x}) + \epsilon = \mathbf{x}^T\mathbf{a} + \epsilon = a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + a_k \cdot x_k + \epsilon,$$

where ϵ follows a certain probability distribution. However, if we want to perhaps model interaction effects of the input, the following would also be a valid function for a linear regression model:

$$\mathbf{y} = f(\mathbf{x}) + \epsilon = a_1 \cdot x_1 x_2 + a_2 \cdot x_3 x_4 + \epsilon$$

There are various approaches to choosing the optimal parameters $\mathbf{a}$; one of the most widely used methods is by maximum likelihood estimation. Given a set $\mathcal{D} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_k}, y_k)\}$ of input-output pairs (usually known as training data), we maximize the likelihood function of the parameters $\mathbf{a}$. How we maximize this function depends on the class of our function f. For this thesis, the exact technique of parameter estimation is not so relevant as we mostly use standard functions built into Python libraries to train our models [5].

Logistic regression

Logistic regression differs from classic regression models in that the output of the model is binary (0 or 1); it is more often seen as a classification algorithm than a regression algorithm. A classification algorithm, as the name suggests, gives an output in the form of a predicted 'class' from a discrete set of predetermined outcomes as opposed to a predicted value in a continuous range. Binary classification problems involve just two classes.

We seek a function f in a manner similar to linear regression, and subsequently pass it through the logistic sigmoid function defined as:

$$\sigma(f) = \frac{1}{1 + \exp(-f)},$$

where $\sigma(f) \in [0, 1]$. The class is then determined by comparing the output to a certain threshold. This output can be seen as a probability of observing an outcome of 1 [5].

Neural network

Neural networks are somewhat more complicated than the regression models discussed previously. A neural network is comprised of layers, with each layer containing a number of so-called activations. We briefly sketch the structure of a neural network with two layers; for input variables $x_1, \ldots, x_D$ we construct M so-called 'activations' a_j with $j = 1, \ldots, M$ of the form:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}$$

Here, (1) corresponds to the first so-called 'hidden layer' of the neural network. The values w_{ji} are known as weights, and b_j as the biases. We transform these activations using an 'activation function' h, yielding input variables for the next layer of the network:

$$z_j = h(a_j)$$

For the variables $z_1, \ldots, z_J$ we then construct K activations a_k with $k = 1, \ldots, K$ of the form:

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + b_{k0}^{(2)}$$

These activations are found in the second layer of the neural network. Again, we use an activation function to determine the model outputs y_k.

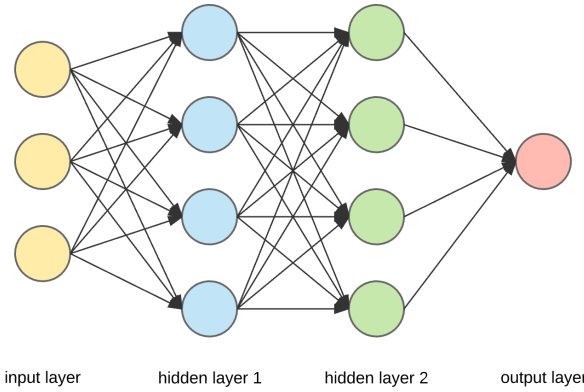

Figure 4.1: A visual representation of a neural network. [6]

There are many different possibilities for activation functions, amount of layers, and number of activations in each layer. Usually this depends on the nature of the input data. Neural network performance can be improved by tuning these parameters. However, as mentioned before, optimizing model performance is not the primary focus of this thesis and thus we will not spend too much time trying to determine the 'perfect' neural network. [7]

Random forest

A decision tree, or a classification tree, is a classification method in which the input is put through a set sequence of binary true/false 'tests'. Through every test, a number of classes is rejected until a leaf node of the tree is reached, associated with a single class which the algorithm returns as an output. The tests are also known as 'splitting criteria'; when we train a decision tree, splitting criteria are generated automatically. However, we do have control over parameters such as the 'depth' of the tree, (number of edges from the root node to the furthest leaf).

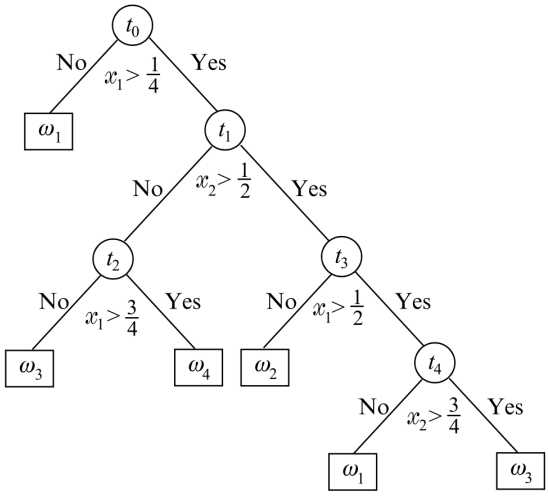

Figure 4.2: A visual representation of a decision tree. [8]

A random forest is a classification algorithm which makes use of multiple decision trees. Multiple training sets are generated from the original data by uniformly sampling with replacement (i.e. bootstrapping), and a decision tree is trained using each set. The class predicted by a 'majority vote' of these decision trees is the output class of the random forest [8].

4.2 Application to C&CG algorithm

Now that we have given some brief mathematical intuition behind a few widely-used machine learning models, we will illustrate how we can implement these models into the C&CG algorithm. In this thesis, we focus on two different ways of implementing a machine learning predictor into the C&CG. The most intuitive way to explain these implementations is through a flowchart diagram. We first summarize the C&CG as presented in Section 2.2 using such a diagram:

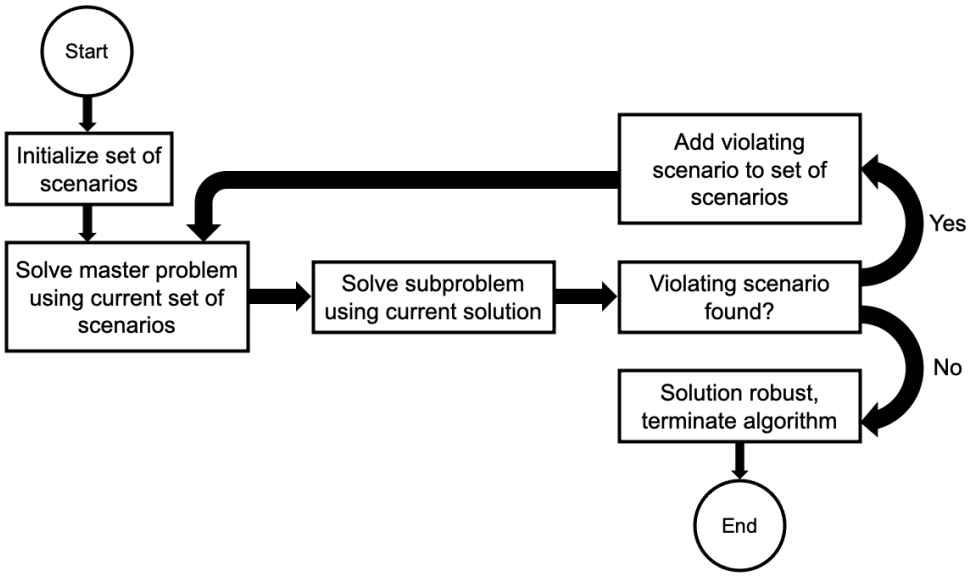

Figure 4.3: A visual representation of the C&CG algorithm.

The motivation behind our attempt to improve the algorithm using ML lies in the step of solving the subproblem. We have mentioned previously that for every instance of the location transportation problem that we examine in this thesis, there is some 'minimal set of important scenarios'. We will denote this by:

$$\mathcal{Z}_I \subset \dot{\mathcal{Z}} = \left\{ \mathbf{z} \in \text{ext}(\mathcal{Z}) : \sum_{c=1}^{C} z_c = \Gamma \right\}$$

The following properties hold for this set:

- Solving the master problem for $\mathcal{Z}_I$ gives us a feasible optimal solution for all realizations of uncertainty, i.e. $\forall \mathbf{z} \in \mathcal{Z}$.
- For $\mathbf{z}_j \notin \mathcal{Z}_I$, solving the master problem for $\mathcal{Z}_I \cup \mathbf{z}_i$ gives the same first-stage decision as when we solve for $\mathcal{Z}_I$.

- For $\mathbf{z}_i \in \mathcal{Z}_I$, the first-stage decision obtained by solving the master problem for $\mathcal{Z}_I \setminus \{\mathbf{z}_i\}$ is infeasible for some $\mathbf{z} \in \mathcal{Z}$.

Essentially, we have that $\mathcal{Z}_I$ is the minimum-size set covering the uncertainty set $\mathcal{Z}$. Clearly, the C&CG algorithm would be most efficient if every violating scenario we find would belong to $\mathcal{Z}_I$, as this would require the algorithm to run for a minimal number of iterations. Furthermore, as we add more violating scenarios to the current uncertainty set in the algorithm, we are also adding more constraints to the master problem and looking for more second-stage solutions, requiring more computations. Thus, it is clear that identifying the important solutions early on is beneficial for the computational efficiency of the C&CG algorithm.

4.2.1 Implementations

In this thesis, we examine two slightly different methods of implementing the ML predictions into the original C&CG algorithm. In both implementations, we run $\dot{\mathcal{Z}}$ through a machine learning classifier to obtain a prediction of whether each scenario is important for the problem or not (i.e. if $\mathbf{z}_i \in \mathcal{Z}_I$ or not). This classification is binary (i.e. each scenario is either important or unimportant); there is no ranking in importance in our implementation.

Implementation A

In the first method (which we will refer to as 'Implementation A'), we firstly determine the set of important scenarios for our problem using a machine learning tool. We then iteratively add these scenarios to our problem until we find a solution that is feasible for the entire uncertainty set. In the case that we do not find a feasible solution using the predicted important scenarios we continue with the 'regular' C&CG algorithm, iteratively solving the master problem and subproblem searching for a violating scenario until a robust solution is found. Note that if our ML model could predict the important scenarios with 100% accuracy, we would never reach this case in this implementation. The process is shown in Figure 4.4:

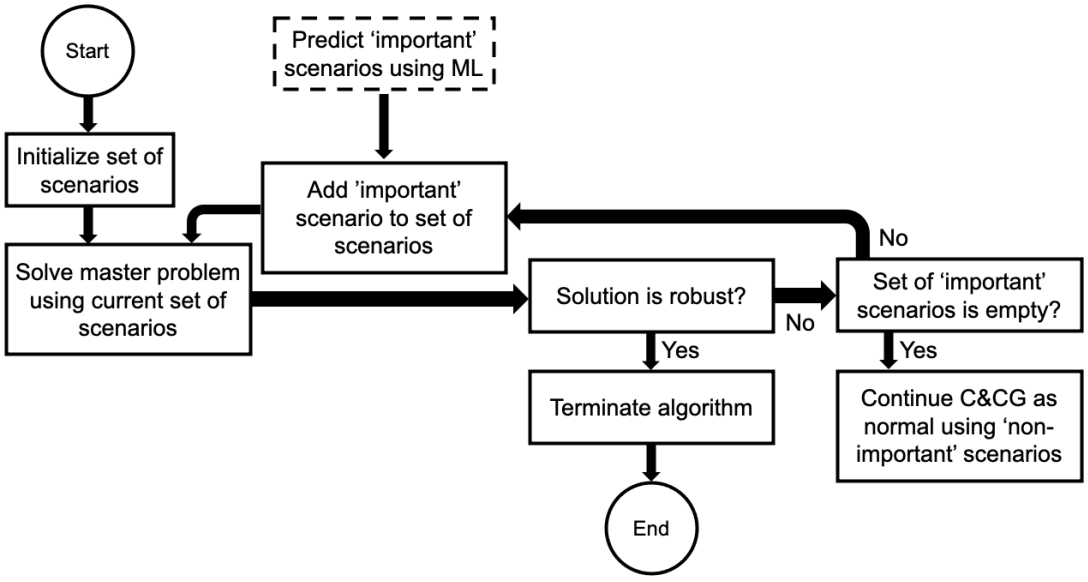

Figure 4.4: Implementation A of our adapted C&CG algorithm with machine learning.

Implementation B

Our second implementation (Implementation B) uses the predicted set of important scenarios in a similar manner. Here, we also look for a robust solution using the predicted set; if this fails, we continue the C&CG as normal. However, there is a small difference in how we select scenarios to add to the problem: in the first stage, instead of just randomly choosing a scenario from the predicted set every iteration, we make use of the subproblem to look for a violating scenario in the predicted set. The implementation is shown in Figure 4.5:

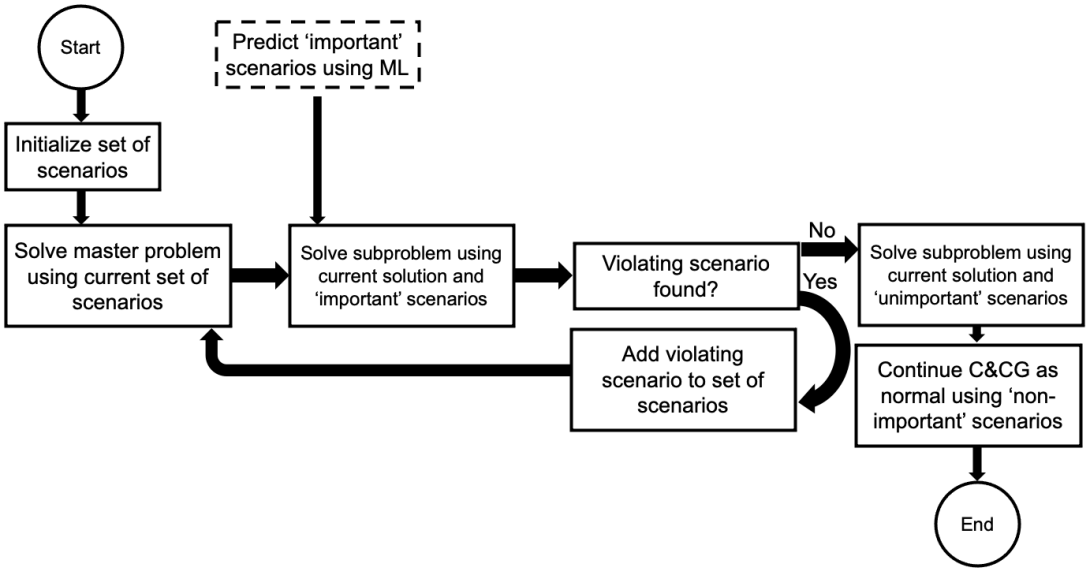

Figure 4.5: Implementation B of our adapted C&CG algorithm with machine learning.

4.2.2 Model training

For a machine learning model to be able to make predictions of important scenarios for our problem, we must first train the model on existing data. In most cases of training ML models, the more data a model has 'seen' the better it will perform in identifying underlying patterns. In this thesis, we build up our set of training data by solving many different instances of the location transportation problem and identifying the important scenarios for each problem by inspection.

In this section, we will explain how we generate our training data and discuss the features of the problem that can be incorporated in our models to predict whether a given scenario is important for a problem.

Data generation

For a given instance of a location transportation problem, our model must take certain features of the problem as an input, and for every scenario in the set $\dot{\mathcal{Z}} = \left\{\mathbf{z} \in \text{ext}(\mathcal{Z}) : \sum_{c=1}^{C} z_c = \Gamma\right\}$ it must return some indication of whether the scenario is in $\mathcal{Z}_I$ for our problem as an output. As mentioned previously, in our implementation we choose to make scenario importance a binary variable (i.e. a scenario is either important or not important to the problem).

Thus, for a problem instance i, we want to generate training data $(\mathbf{x}^i, y^i)$ where $\mathbf{x}^i = [x_1^i, \ldots, x_n^i]$

is a vector of n input features for the problem instance (which we will discuss in the next section) and $y^i \in \{0, 1\}$.

A single problem instance is defined by the variables and parameters that make up the objective function and constraints. In building up our training set, there are certain parameters that we keep constant (in order to generate the same type of problem), and some that we vary (in order to create different instances of the same type). These parameters are listed below:

Variable	Description	Range
S	Number of facilities in the problem	$\{5\}$
C	Number of customers in the problem	$\{10\}$
$\overline{d}_c$	Nominal demand of customer c	$[10, 500]$
δ_c	Max. demand deviation of customer c	$a \cdot \overline{d}_c$, with $a \in [0.1, 0.5]$
f_s	Fixed cost of opening facility s	$[1, 10]$
v_s	Variable unit cost of production at facility s	$[0.1, 1]$
$t_{s,c}$	Unit transport cost from facility s to customer c	$[0, 10]$
Γ	Max. budget for the uncertainty set	$\gamma\% \cdot C$, with $\gamma\% \in \{0.8, 0.9\}$

Table 4.1: Variables used to generate an instance of the location transportation problem.

We keep the number of customers C and facilities S constant throughout all our problem instances. Variables with values in a certain range (i.e. customer demand/deviation and costs) are uniformly sampled within the range. Furthermore, we examine 2 cases for values of $\gamma\%$ in this thesis. This value influences the size of $\dot{\mathcal{Z}}$, so it is interesting to investigate how the number of important scenarios differs as we vary this parameter.

Using the data above, we can randomly generate a large number of instances of the location transportation problem. For each instance, we must identify the minimal set of important scenarios. This can be done by solving the master problem using all scenarios in $\dot{\mathcal{Z}}$ and checking one-by-one whether the scenario is important for the problem. This is done by comparing the objective values before and after removing the scenario from the master problem. If the value is unchanged after removing the scenario, this implies that the first stage decision is feasible for both sets and the scenario is thus not important to the problem as a whole (and can be removed). We describe this process of identifying important scenarios in Algorithm 2:

Algorithm 2: Identification of important scenarios for a given problem.

$\mathcal{Z}_I = \emptyset$

Solve the MP using $\dot{\mathcal{Z}} = \left\{\mathbf{z} \in \text{ext}(\mathcal{Z}) : \sum_{c=1}^{C} z_c = \Gamma\right\}$

Set O_{init} to be the objective value of the MP

for $\mathbf{z} \in \dot{\mathcal{Z}}$ **do**

    Solve the MP using $\dot{\mathcal{Z}} \setminus \{\mathbf{z}\}$

    Set O_{new} to be the objective value of the MP

    if $O_{new} < O_{init}$ **then**

        Add z to $\mathcal{Z}_I$

Model input

In the manner described in the previous section, we can generate arbitrarily many 'problem-to-important-scenario' pairs to build up our training dataset. For every problem, we can label the scenarios in the set $\dot{\mathcal{Z}}$ as important/not important. All that remains to build our training set, is to extract certain features of the problem instances to use as input variables for the machine learning model.

There are many possibilities and combinations of problem features to incorporate in our model. In this thesis, we will focus on the customer demand data for a given problem. Intuitively, the nominal customer demand $\overline{d}_c$ and the maximum demand deviation δ_c may be related to the importance of scenarios in the uncertainty set, since the uncertainty set $\mathcal{Z}$ contains data regarding the customer demand:

$$d_c = \overline{d}_c + z_c\delta_c,$$

where $\mathbf{z} = (z_1, \ldots, z_C) \in \mathcal{Z}$. As seen in the previous section, $\overline{d}_c$ and δ_c are randomly sampled in every problem instance. Thus, for every problem instance we generate, we extract these values for every customer. For $|\dot{\mathcal{Z}}| = K$, our training data for one problem instance will have the following form (using example values for demand):

Scenario	$\overline{d}_1$	δ_1	...	$\overline{d}_C$	δ_C	**Important**
1	236.71	108.78	...	240.73	110.43	0
2	236.71	108.78	...	240.73	110.43	1
3	236.71	108.78	...	240.73	110.43	0
⋮	236.71	108.78	...	240.73	110.43	⋮
$K-1$	236.71	108.78	...	240.73	110.43	1
K	236.71	108.78	...	240.73	110.43	0

Table 4.2: Training data for an instance of the location transportation problem.

Note that *Scenario* is a categorical variable, not a numerical variable. The $\overline{d}_i$ and δ_i variables are numerical, and *Important* is a binary/boolean variable.

Training phase

By extracting data in the manner above for many ($N = 5000$, in our case) randomly generated instances of the location transportation problem, we obtain a useful training set for fitting a classification model (in our case, we make use of logistic regression, a neural network and a random forest). We use the Python library *Scikit-Learn* for training the models, which has built-in methods for optimizing the model parameters.

We first split the training data into a training set (80% of the data) and a test set (20% of the data), and fit the model using the training set. We obtain a model that, when given a scenario from $\dot{\mathcal{Z}}$ and a set of input variables, returns a probability $p \in [0, 1]$ that the scenario is important (according to the model).

In this thesis, our model is technically built up from 'smaller' classifiers, one for each scenario in $\dot{\mathcal{Z}}$. We obtain this by separating the data by scenario and training a model for each scenario. Then, depending on the input scenario, the corresponding classifier is used.

In order to be able to use the model to actually classify scenarios, we must choose a threshold t such that the scenario is only classified as important if $p > t$. We determine this threshold using the test set that we obtained from the training data, and some commonly-used metrics in machine learning. Consider the following matrix (known as a 'confusion matrix') [8]:

	Predicted Class = True	**Predicted Class = False**
Actual Class = True	True Positive (TP)	False Negative (FN)
Actual Class = False	False Positive (FP)	True Negative (TN)

Table 4.3: Confusion matrix for a classification model.

Two commonly used metrics for classification models are precision and recall. These are defined as follows [8]:

$$\begin{aligned} \text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\ \text{Recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \end{aligned}$$

There is a slight difference between these metrics. Suppose we run our scenario classification model on a set of input data. Precision is the ratio of scenarios that we correctly classified as important to the total number of scenarios that we classified as important. Recall is the ratio of scenarios that we correctly classified as important to the total number of important scenarios. A good precision score (close to 1) indicates that we can be confident that a scenario in our predicted important set is actually important. A good recall score indicates that we can be confident that if a scenario is actually important, it is in our predicted important set.

Note that we could technically obtain a recall score of 1 by simply labelling all scenarios as important. This would, however, negatively impact our precision score due to the number of false positives. We often have some trade-off in precision and recall, which motivates a weighted average of these two metrics, known as the F1-score [8]:

$$\text{F1-score} = \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}$$

Once we have trained our scenario classifier using the training set, we run the input variables from the test set through the model and compare the model prediction to the true output in the test set. By calculating the F1-score for different thresholds, we look for $t \in [0, 1]$ such that the F1-score is maximised and take this to be our model's threshold parameter.

4.2.3 Model testing

Once our models have been trained and thresholds have been calculated, it remains to investigate whether the 'adapted' C&CG algorithm using ML performs better than the 'regular' C&CG algorithm. We do this by generating many ($N = 1000$) instances of the location transportation problem and running both algorithms on the problem. We can then extract the number of scenarios that have been added to the uncertainty set once the algorithm terminates and examine whether the ML-C&CG algorithm indeed needs fewer scenarios on average.

The results are presented and discussed in the next chapter.

5 | Results

In this section, we present the results obtained from training various machine learning models to predict important scenarios for the C&CG algorithm applied to the location transportation problem with 5 facilities (S) and 10 customers (C), for both $\gamma\% = 0.9$ and $\gamma\% = 0.8$. We examine the following models:

- Logistic regression classifier (LR)
- Neural network (NN)
- Random forest (RF)

We test each model in the two different implementations described in the previous section.

5.1 Model performance

Before going into the results of our 'adapted' C&CG algorithm, we briefly present the models that we use for making predictions. We provide some performance metrics obtained from test datasets and discuss how we expect the models to perform in practice.

The performance metrics are presented in a so-called 'classification report' obtained from the *Scikit-Learn* package [9], which contains various metrics regarding the performance of our models on the test set. We have already defined precision, recall and F1-score; aside from these, it is useful to define the following terms [8]:

- *Support* of a class is the number of samples in the test set that lie in that class.
- *Accuracy* is the proportion of correctly classified instances over all instances, i.e.:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- *Macro average* is the average of multiple scores.
- *Weighted average* is the average of multiple scores weighted according to their support.

For our models, the metrics related to the *1 (Important)* class are the most important; ideally, we want every important scenario to be classified as such by our model (high recall) whilst minimizing the number of false positives. Looking at the F1-score for this class will give a good idea of the model performance for our purposes.

The following metrics are obtained for the logistic regression classifier:

	Precision	**Recall**	**F1-Score**	**Support**
0 (Unimportant)	0.89	0.84	0.86	7976
1 (Important)	0.49	0.61	0.54	2024
Accuracy			0.79	10000
Macro Average	0.69	0.72	0.70	10000
Weighted Average	0.81	0.79	0.80	10000

Table 5.1: Classification report for LR model with $\gamma\% = 0.9$.

	Precision	**Recall**	**F1-Score**	**Support**
0 (Unimportant)	0.97	0.94	0.95	42426
1 (Important)	0.32	0.46	0.38	2574
Accuracy			0.91	45000
Macro Average	0.64	0.70	0.67	45000
Weighted Average	0.93	0.91	0.92	45000

Table 5.2: Classification report for LR model with $\gamma\% = 0.8$.

The following metrics are obtained for the neural network (with 2 hidden layers of size 50 and 200 epochs):

	Precision	**Recall**	**F1-Score**	**Support**
0 (Unimportant)	0.88	0.74	0.80	7976
1 (Important)	0.37	0.60	0.45	2024
Accuracy			0.71	10000
Macro Average	0.62	0.67	0.63	10000
Weighted Average	0.77	0.71	0.73	10000

Table 5.3: Classification report for NN model with $\gamma\% = 0.9$.

	Precision	**Recall**	**F1-Score**	**Support**
0 (Unimportant)	0.96	0.91	0.93	42426
1 (Important)	0.21	0.39	0.27	2592
Accuracy			0.88	45000
Macro Average	0.58	0.65	0.60	45000
Weighted Average	0.92	0.88	0.90	45000

Table 5.4: Classification report for NN model with $\gamma\% = 0.8$.

The following metrics are obtained for the random forest classifier:

	Precision	Recall	F1-Score	Support
0 (Unimportant)	0.89	0.87	0.88	7976
1 (Important)	0.54	0.58	0.56	2024
Accuracy			0.81	10000
Macro Average	0.71	0.73	0.72	10000
Weighted Average	0.82	0.81	0.82	10000

Table 5.5: Classification report for RF model with $\gamma\% = 0.9$.

	Precision	Recall	F1-Score	Support
0 (Unimportant)	0.96	0.95	0.96	42426
1 (Important)	0.34	0.39	0.37	2574
Accuracy			0.92	45000
Macro Average	0.65	0.67	0.66	45000
Weighted Average	0.93	0.92	0.92	45000

Table 5.6: Classification report for RF model with $\gamma\% = 0.8$.

Judging purely by these metrics, the logistic regression and random forest classifiers are expected to perform similarly at predicting important scenarios. The neural network is expected to perform slightly worse than the other two models. We judge this by looking at the F1-score as discussed previously.

Furthermore, we see that the scores for $\gamma\% = 0.9$ are in general better than for $\gamma\% = 0.8$ when it comes to the *1 (Important)* class, but worse for the *0 (Unimportant)* class. We also see that scores for the *0 (Unimportant)* class are generally higher than for the *1 (Important)* class. This is most likely related to the support for these classes. We see that for $\gamma\% = 0.9$, roughly 20% of the test set is labeled as *1 (Important)* and 80% as *0 (Unimportant)*. For $\gamma\% = 0.8$, the split is roughly $5\% - 95\%$. We are dealing with an unbalanced data set in favour of the *0 (Unimportant)* class, which makes it more difficult to train our models for the other class. This explains the differences.

Note that for both values of $\gamma\%$, there are on average just $2 - 3$ important scenarios for a given problem instance. We have that $|\dot{\mathcal{Z}}| = 10$ for $\gamma\% = 0.9$, and $|\dot{\mathcal{Z}}| = 45$ for $\gamma\% = 0.8$. It is interesting that, even though we observe this difference in the uncertainty set, the average number of important scenarios for a problem is roughly equal. This leads to the imbalances in support that we mentioned earlier.

5.2 Results of applying ML to C&CG

Now that we have seen that most of our models perform quite decently at predicting important scenarios for unseen problems, we present the result of implementing these models into the C&CG algorithm.

The following tables contain the results of running the algorithm 'normally', as well as with each ML model (in both implementations) for two different values of $\gamma\%$.

	Implementation A			Implementation B		
	Avg. # scenarios (regular)	Avg. # scenarios (ML)	Difference	Avg. # scenarios (regular)	Avg. # scenarios (ML)	Difference
Logistic Regression	5.865	5.014	0.851 (14.51%)	5.903	4.988	0.915 (16.11%)
Neural Network	5.865	5.389	0.476 (8.12%)	5.903	5.426	0.477 (8.08%)
Random Forest	5.865	5.033	0.832 (14.19%)	5.903	5.102	0.801 (13.57%)

Table 5.7: Results of applying ML to C&CG for $\gamma\% = 0.9$, with $N = 1000$.

	Implementation A			Implementation B		
	Avg. # scenarios (regular)	Avg. # scenarios (ML)	Difference	Avg. # scenarios (regular)	Avg. # scenarios (ML)	Difference
Logistic Regression	24.041	20.281	3.76 (15.64%)	22.449	17.737	4.712 (20.99%)
Neural Network	24.041	20.786	3.255 (13.54%)	22.449	19.3	3.149 (14.03%)
Random Forest	24.041	20.949	3.092 (12.86%)	22.449	17.659	4.79 (21.33%)

Table 5.8: Results of applying ML to C&CG for $\gamma\% = 0.8$, with $N = 1000$.

In the results presented above, we first indicate the average number of scenarios that we add to our problem in running the C&CG algorithm on $N = 1000$ newly generated test instances of the location-transportation problem. This value is found under *Avg. # scenarios (regular)*. In generating our results, we generated separate problems for each implementation (and $\gamma\%$, of course). We see that for $\gamma\% = 0.9$, nearly 6 of the 10 scenarios in $\dot{\mathcal{Z}}$ are added to the problem on average. For $\gamma\% = 0.8$, this is around 24 of the 45 scenarios.

The next column, *Avg. # scenarios (ML)*, indicates the average number of scenarios added to our problem in the C&CG when predicting scenarios using an ML model. The *Difference* column, as the name suggests, is the difference between these two values.

5.3 Discussion of results

For $\gamma\% = 0.9$, we see that for both implementations the logistic regression classifier performs best, followed closely by the random forest classifier. When applying these models to the C&CG in the manner described in this report, we see that on average the set of scenarios we end up with is almost a whole scenario smaller than when just running the regular C&CG. Though the difference is not extremely significant, these results indicate that predicting the right scenarios early can indeed improve the runtime of the algorithm. The neural network also gives positive results, but to a lesser degree than the other two models. Furthermore, there seems to be no significant difference in results between the two implementations.

For $\gamma\% = 0.8$, we also observe positive results. We see that in Implementation A, the logistic

regression classifier performs best, then the neural network, closely followed by the random forest. In Implementation B, the logistic regression classifier and random forest classifier perform roughly equally well again, with the neural network scoring slightly lower. Furthermore, we see that in Implementation B the average number of scenarios when using the predictors is quite a bit lower than in Implementation A, though this may also be related to the problems generated for these results (as the average number of scenarios in the 'regular' C&CG is also lower).

5.3.1 Model coefficients

In order to gain some more insight into how the models predict the important scenarios, it is interesting to look at the parameters of the trained models. Specifically, we will examine the coefficients of the logistic regression classifiers.

As mentioned previously, our classifier is made up of multiple 'sub-models'; that is, for every scenario in $\dot{\mathcal{Z}}$, we train a logistic regression classifier such that when we input customer demand data into the classifier, we receive a prediction whether the scenario corresponding to the classifier is important to the problem. Thus, for $\gamma\% = 0.9$, our model is made up of 10 different classifiers since $|\dot{\mathcal{Z}}| = 10$. Similarly, for $\gamma\% = 0.8$, the model is composed of 45 classifiers since $|\dot{\mathcal{Z}}| = 45$ for this problem.

To better interpret the coefficients, it is also useful to mention what the scenarios in $\dot{\mathcal{Z}}$ are. Recall that for a given problem, we have an 'uncertainty budget' $\Gamma = \gamma\% \cdot C$. Thus, for $\gamma\% = 0.9$ and $C = 10$, we have $\Gamma = 9$. Thus, we have:

$$\dot{\mathcal{Z}} = \left\{ \mathbf{z} \in \text{ext}(\mathcal{Z}) : \sum_{c=1}^{C} z_c = 9 \right\}$$

In other words, $\dot{\mathcal{Z}}$ is the set of extreme points of $\mathcal{Z}$ of which the coordinates add up to 9 (for $\gamma\% = 0.9$. This means that the uncertainty set contains the following scenarios:

$$\begin{array}{lcl}
\text{Scenario 0} & = & [0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1] \\
\text{Scenario 1} & = & [1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1] \\
\text{Scenario 2} & = & [1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1] \\
\text{Scenario 3} & = & [1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1] \\
\text{Scenario 4} & = & [1\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1] \\
\text{Scenario 5} & = & [1\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1] \\
\text{Scenario 6} & = & [1\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1] \\
\text{Scenario 7} & = & [1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1] \\
\text{Scenario 8} & = & [1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1] \\
\text{Scenario 9} & = & [1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0]
\end{array}$$

We see that, in scenario i, the demand deviation z_c for the corresponding customer $c = i$ is equal to 0. Note that $\dot{\mathcal{Z}}$ is technically not an ordered set, but for this purpose it is useful to enumerate the set in this manner.

The following two plots display the coefficients of the logistic regression classifiers for two scenarios in $\dot{\mathcal{Z}}$:

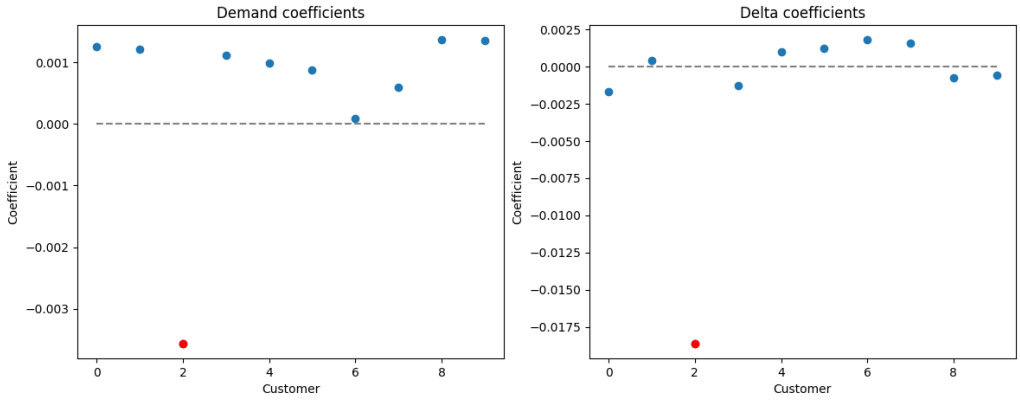

Figure 5.1: Coefficients of LR classifier for scenario 2, for $\gamma\% = 0.9$.

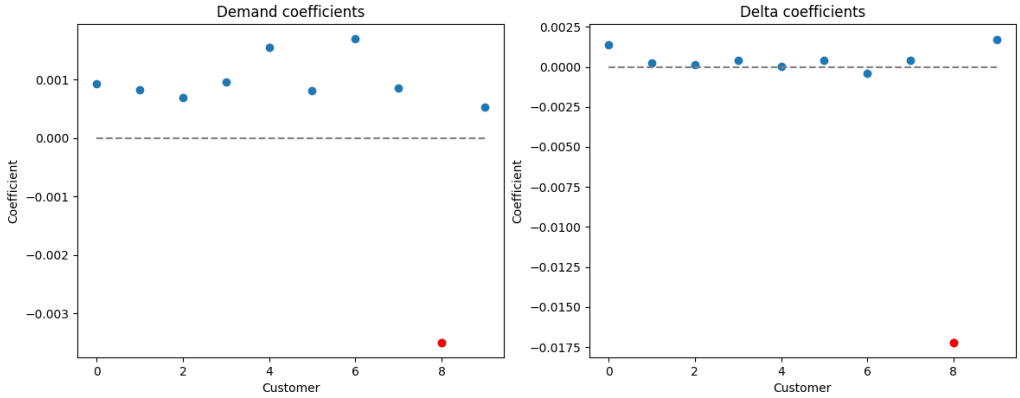

Figure 5.2: Coefficients of LR classifier for scenario 8, for $\gamma\% = 0.9$.

In the figures above, we have colored the coefficient corresponding to the demand data for customer i in red. We see that, for these two classifiers, the magnitude of this coefficient is significantly greater than that of the remaining coefficients. Furthermore, this coefficient is negative whilst the rest are positive (or close to zero). For the other classifiers, we see a similar pattern (though we omit the plots in this thesis). Thus, intuitively, the predicted importance of scenario i is largely influenced by the d_c and δ_c data for the problem, for customer $c = i$. The negative sign indicates that if these values are large, the scenario is less likely to be predicted as important.

For $\gamma\% = 0.8$, we have a similar pattern in the coefficients. We can see this by plotting the coefficients for two scenarios; in this case, we take 20 and 27. These scenarios are given below:

$$\begin{aligned} \text{Scenario } 20 &= [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1] \\ \text{Scenario } 27 &= [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1] \end{aligned}$$

The customers for which $z_c = 0$ in these scenarios correspond to the highlighted coefficients in the plots in Figure 5.3 and Figure 5.4.

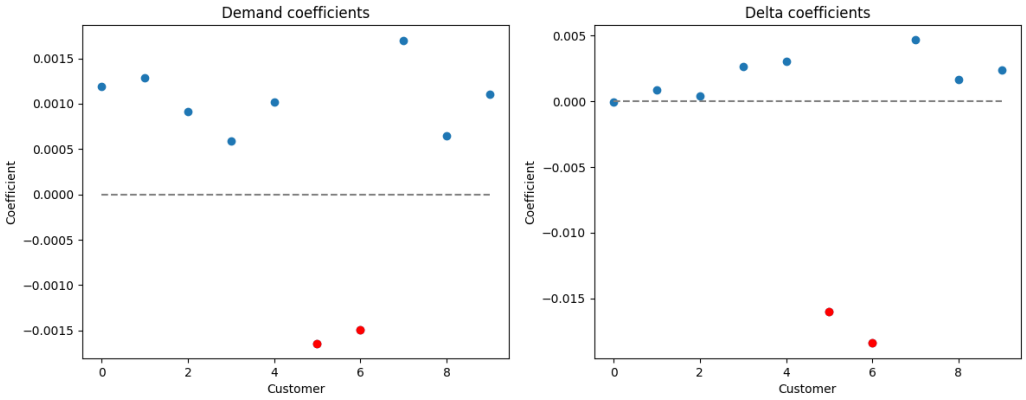

Figure 5.3: Coefficients of LR classifier for scenario 20, for $\gamma\% = 0.8$.

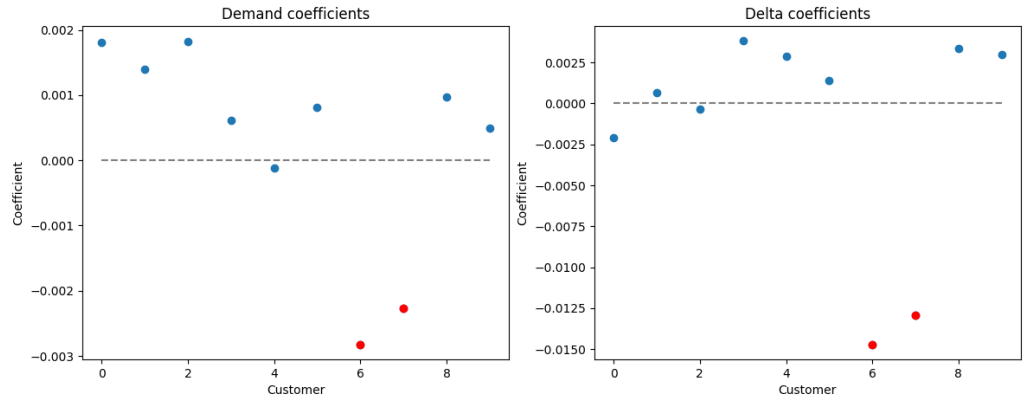

Figure 5.4: Coefficients of LR classifier for scenario 27, for $\gamma\% = 0.8$.

5.3.2 Improvements to model

We have seen that our three machine learning models perform quite well at classifying important scenarios for a given problem based on customer demand data and, when implemented into the C&CG, improve the runtime of the algorithm. We have also shown that some logic/sense can be found in the parameters of the trained models. Of course, there are many aspects that could still be improved with regards to machine learning.

Firstly, as mentioned previously, the dataset that we use to train our models is rather imbalanced (especially for problems with $\gamma = 0.8$). This is due to the fact that the minimal set of important scenarios is quite small (on average 2-3 scenarios) compared to the complete set $\dot{\mathcal{Z}}$ from which we want to predict scenarios. There are various methods that are often used to combat an imbalanced dataset, such as resampling. This could entail removing samples from the overrepresented class in the training set, or generating 'synthetic' data that resembles data from the underrepresented class. We won't focus on the details of these methods in this thesis, but for further research this is definitely an aspect to look at as this would improve the support of the important scenarios in our training set. This, in turn, can potentially improve the performance metrics of our models.

Another area on which the models could potentially be improved is by examining the training

parameters more closely. In this thesis, for the logistic regression and random forest classifiers, we used the standard settings in the *Scikit-Learn* package [9] for these models in the training phase. For the neural network, we only specified the hidden layer sizes. This is because the focus of this thesis lies much more on the optimization problem than on the details of the machine learning models. However, given more time and expertise, it could be interesting to tune the various parameters of the models to see if the classifiers can be made to perform significantly better. Furthermore, generating more instances of the training data than we did in this thesis could also be a way to improve model performance. The data generation was quite time-consuming, but given more time and possibly better computational resources this would improve performance.

One small point that could improved with regards to model complexity is the fact that we trained multiple classifiers and 'combined' these into one model. Ideally, we would just train one model using the scenario as a (categorical) input as opposed to splitting the data by scenario and training a separate model on each dataset. This was (shortly) attempted in this thesis, but this didn't give very positive results. However, intuitively, it should be possible to get the same performance as our final models using just one model (perhaps using interaction effects between scenarios and demand data).

Lastly, it could be interesting to look at a completely different type of training data. For this thesis, we chose to look at customer demand data for a given problem since the uncertainty in the problem is closely related to customer demand. However, an instance of the location transportation problem has many different variables that could be incorporated into our predictive model, such as costs of production and transport. We could also look at data obtained during the process of running the C&CG algorithm, such as the last found second-stage decision or the current optimal first-stage decision. Intuitively, these may contain information about where the algorithm should 'look' in the next iteration. It may even be possible to study interaction effects of different variables and parameters. All in all, there are more than enough aspects that can still be explored.

6 | Conclusion

The main research question that we aimed to to answer in this thesis was: 'How can machine learning models be used to improve the efficiency of the C&CG algorithm?'. To achieve this, we first introduced multi-stage (and specifically two-stage) optimization problems, building up from linear programming models and robust counterparts to such problems. We presented the column-and-constraint generation algorithm that can be used to solve a two-stage optimization problem and discussed some of its drawbacks. We then explained the location transportation problem, a specific real-life problem to which the C&CG can be applied, and introduced the concept of the 'important scenarios' for such a problem.

Next, we briefly covered the mathematical intuition behind a few machine learning models and showed how applying a predictive ML classifier to the C&CG algorithm might help to improve its runtime by reducing the number of uncertain scenarios the algorithm needs to 'look at'. We explained how three different models - a logistic regression classifier, a neural network and random forest - were trained to predict these scenarios. Subsequently, the results were presented and discussed.

Overall, as previously stated, the results obtained from incorporating machine learning into the optimization algorithm were positive. We saw a small improvement in the average runtime of the algorithm due to the prediction of important scenarios, indicating that there is some underlying pattern in the customer demand data for the location transportation problem as well as a link between this data and the right scenarios to add to the problem. We explored this relationship by visualizing the coefficients of the logistic regression classifier, which seemed to perform best overall out of the three models.

All in all, this thesis is an interesting demonstration of how combining two disciplines in mathematics (optimization and machine learning) can produce useful results. There is, of course, still a great deal to explore in the overlap of these two fields. In the previous section, we mentioned just a few directions in which further research can be conducted for this project alone, but there are of course many more possible ways in which optimization algorithms can be boosted with machine learning.

Lastly, since we were dealing mostly with 'small' instances of the location transportation problem in this project (i.e. few customers/facilities), the results aren't too ground-breaking; however, it would definitely be interesting to see whether applying machine learning techniques to larger optimization problems in industry applications could significantly save resources in large companies and businesses where optimization techniques are heavily utilized.

Bibliography

[1] Jean-Michel Réveillac. Linear Programming. In *Optimization Tools for Logistics*, pages 209–237. Elsevier, 2015.

[2] A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25(1):1–13, 1999.

[3] B. Zeng and L. Zhao. Solving two-stage robust optimization problems using a column-and-constraint generation method. *Operations Research Letters*, 41(5):457–461, 2013.

[4] Stephane Caron. Pypoman: Python module for polyhedral geometry.

[5] Marc Deisenroth. *Mathematics For Machine Learning*. Cambridge University Press, Cambridge, United Kingdom New York, NY, 2020.

[6] Shadab Hussain. Building a convolutional neural network: Male vs female, Apr 2020.

[7] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[8] Sergios Theodoridis. Classification: A Tour of the Classics. In Sergios Theodoridis, editor, *Machine Learning*, pages 275–325. Academic Press, Oxford, 2015.

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

A | Python Code

Most of the Python code used for this thesis can be found on GitHub:

`https://github.com/boazvdv/CCG-Thesis`

Note this is purely for illustrative purposes and not all files have been uploaded to GitHub (e.g. data, results, models) which may cause errors when running the code.