Optimization in Data Analysis Project

R8: Mixed Integer Programming with $p >> n$

Hubert Ruczyński, Kinga Ułasik

June 2023

Faculty of Mathematics and Information Science
WARSAW UNIVERSITY OF TECHNOLOGY

Contents

1 Theoretical Introduction

1.1 Introduction

Linear regression is a fundamental statistical technique used to analyze and model the relationship between a target variable and features from observed data. It serves as a powerful tool for understanding and predicting trends, making informed decisions, and gaining insights from data. In its simplest form, the linear regression model can be represented as:

$$y = X_1\beta_1 + \cdots + X_p\beta_p + \varepsilon \tag{1}$$

where y represents the dependent variable, $X_1, \ldots X_p$ denote features from model matrix $X \in \mathbb{R}^{nxp}$, $\beta_1 \ldots \beta_p$ are the coefficients associated with each feature, and ε represents the random error term. The goal is to estimate the coefficients β that best describes the relationship between the dependent variable and the independent variables.
Linear regression finds applications in various fields, including but not limited to healthcare, medicine, economics, finance, marketing, and social sciences.
Understanding linear regression is essential for anyone involved in data analysis and modeling. By utilizing mathematical formulation and applying it to practical scenarios, linear regression empowers us to make predictions, uncover hidden patterns, and gain valuable insights from data.

1.2 Best subset selection problem

Throughout this report, we will dwell deeper into the concept of best subset selection method using modern optimization methods, specifically mixed integer optimization (MIO). The focus will be put on the case where $p >> n$, so when, data consist of way more features than observations.

The Mixed Integer Quadratic Optimization (MIO) problem is a powerful framework that has made significant advancements in the past twenty-five years. This report explores the application of MIO formulations in the context of the best subset problem and its connection with compressed sensing literature. The proposed approach formulates the best subset selection problem as a quadratic optimization task, aiming to minimize the objective function $\alpha^T Q\alpha + \alpha^T a$ subject to linear constraints $A\alpha \leq b$.
M(I)QP (Mixed Integer Quadratic Programming) is a specific variant of MIO. M(I)QP formulations allow for the inclusion of both continuous and discrete decision variables while considering quadratic objective functions and linear constraints. M(I)QP formulations are often used to address complex regression problems where variable selection and binary decisions are required, and such was our case.

The best subset selection problem aims to determine the optimal subset of features from the model matrix X to include in a linear regression model. One commonly used criterion in the best subset selection is the least squares criterion, which aims to minimize the sum of squared differences between the predicted values and the actual values of the dependent variable. Other criteria, such as adjusted R-squared, AIC (Akaike Information Criterion), or BIC (Bayesian Information Criterion), can also be utilized to strike a balance between model fit and complexity. In this project, we focused on the classic best subset problem formulation with the least squares criterion while selecting only k feature from model matrix X. The problem can be represented as:

$$\min_{\beta} \frac{1}{2}||y - X\beta||_2^2 \quad \text{subject to} \quad ||\beta||_0 \leq k \tag{2}$$

As was mentioned before, our work focuses on cases where $p >> n$ in model matrices $X \in \mathbb{R}^{nxp}$.

The majority of our work is based on the article [Bertsimas et al., 2015]. We tried our best to reproduce the implementation presented in the paper and conduct proper experiments that put our ideas to the test.

1.3 Existing solutions

This section provides a concise overview of existing solutions to the best subset problem, which involves selecting the most informative variables for a given task. By examining selected approaches from the literature, we could gain insights into the diverse strategies employed in subset selection. We will briefly go through four of the most popular feature selection methods: Lasso, Ridge, BORUTA, and Mutual Information.

Lasso (Least Absolute Shrinkage and Selection Operator) [Tibshirani, 2018] is a feature selection technique in linear regression that combines variable selection and regularization. It minimizes the sum of squared errors while adding a penalty term based on the absolute values of the coefficients. The objective function is:

$$\min_{\beta} \frac{1}{2}||y - X\beta||_2^2 + \lambda||\beta||_1 \tag{3}$$

The λ parameter controls the level of regularization, promoting sparsity and selecting only the most important features in the model.

Ridge regression [Hoerl and Kennard, 2000] is a feature selection technique that applies regularization to linear regression models. It addresses multicollinearity and stabilizes the model by adding a penalty term based on the squared values of the regression coefficients. The objective function of Ridge regression can be expressed as:

$$\min_{\beta} \frac{1}{2}||y - X\beta||_2^2 + \lambda||\beta||_2^2 \tag{4}$$

λ is again the regularization parameter controlling the strength of the penalty. Ridge regression encourages smaller but non-zero coefficients, allowing all features to contribute to the model while reducing their potential influence.

BORUTA [Kursa and Rudnicki, 2010] is a feature selection algorithm that utilizes a random forest-based approach to identify the most important features in a dataset. It compares the importance of original features with that of randomly created shadow features. The algorithm assigns a tentative or confirmed status to each feature based on statistical tests. The BORUTA feature selection process involves three steps: randomization, importance comparison, and selection. It aims to identify the truly relevant features by considering their relative importance in comparison to the shadow features and statistical significance.

Mutual information (MI) [Sulaiman and Labadin, 2015] is a feature selection technique that measures the dependency between a target variable and each feature in a dataset. It quantifies the amount of information that one variable contains about the other. In the context of feature selection, MI calculates the relevance of features by assessing their information gain with respect to the target variable. The mathematical formula for Mutual Information can be expressed as:

$$M(I, Y) = \sum_{x \in X} \sum_{y \in Y} P(x, y) \log_2 \left(\frac{P(x, y)}{P(x)P(y)} \right) \tag{5}$$

where X and Y represent the feature and target variable, respectively, and $P(X, Y)$, $P(X)$, and $P(Y)$ denote the joint and marginal probabilities. By selecting features with higher MI scores, one can identify the most informative variables for the target prediction task.

1.4 Problem formulations

1.4.1 Specification of parameters

The following subsections will contain information about other various formulations of the main problem. In some of them, values $M_U, M_l, M_U^{\zeta}, M_l^{\zeta}$ appear as problem specific parameters. Generally, they act as upper bounds to certain constraints, to be precise:

$$
\begin{aligned}
||\beta||_\infty &\leq M_U, \\
||\beta||_1 &\leq M_l, \\
||X\beta||_\infty &\leq M_U^{\zeta}, \\
||X\beta||_1 &\leq M_l^{\zeta}.
\end{aligned}
$$

These parameters could be chosen by hand with intuition, but the authors propose a purely data-driver method via convex quadratic optimization to calculate these variables.
Let's firstly define UB as the difference between real y values and the prediction of y achieved by using formula 2.4 described in section 1.4.6. Then we define:

$$
\begin{aligned}
&v_i^+ := \max_\beta \langle x_i, \beta \rangle \qquad && v_i^- := \min_\beta \langle x_i, \beta \rangle \\
\text{s.t}\quad &\frac{1}{2}||y - X\beta||_2^2 \leq UB \qquad \text{s.t}\quad && \frac{1}{2}||y - X\beta||_2^2 \leq UB
\end{aligned}
$$

for every $i = 1, \ldots, n$. Additionally:

$$
v_i = \max\{|v_i^+|, |v_i^-|\}
$$

That leads to obtaining $\max_i v_i$ and $\sum_i v_i$ as simple upper bounds:

$$
\begin{aligned}
||X\beta||_\infty &\leq \max_i v_i \\
||X\beta||_1 &\leq \sum_i v_i
\end{aligned}
$$

Thus values $\max_i v_i$ and $\sum_i v_i$ can act as (respectfully) M_U^{ζ} and M_l^{ζ} parameters in $p >> n$ case.

To calculate the remaining bounds we first ought to acquire approximations of β from the warmstart algorithm from section 1.4.2. Let $\hat{\beta}_{hyb}$ be that approximation. Then:

$$
\begin{aligned}
M_U &:= \tau ||\hat{\beta}_{hyb}||_\infty \\
M_l &:= k M_U
\end{aligned}
$$

where τ is a multiplier greater than 1.

1.4.2 Warmstarts

Warmstarts refer to initializing an algorithm using a simpler solution as a starting point for the main optimization problem. Instead of starting the optimization process from scratch, warmstarts involve using a simpler and quicker method to use the results as the initial values for the main algorithm.

Let's first define the general problem:

$$\min_{\beta} g(\beta) \quad \text{subject to} \quad ||\beta||_0 \leq k \tag{6}$$

where $g(\beta) \geq 0$ is convex and has Lipschitz continuous gradient.

Also, let's define $H_k(c)$ as a function that returns the k largest in absolute value elements from $c \in \mathbb{R}^p$ and sets the rest to zero. The authors propose an algorithm to find a stationary point to the general problem 6:

1. Initialize with $\beta_1 \in \mathbb{R}^p$ such that $||\beta_1||_0 \leq k$.
2. For $m \geq 1$, apply:
$$\beta_{m+1} \in H_k\bigg(\beta_m - \frac{1}{L}\nabla g(B_m)\bigg). \tag{7}$$
where taking β_1 as β_m, L and ε being parameters, $g(\beta) = \frac{1}{2}||y - X\beta||_2^2$ and $\nabla g(\beta) = -X'(y - X\beta)$
3. Repeat step 2, until $g(\beta) - g(\beta_{m+1}) \leq \varepsilon$

After using the above algorithm we obtain satisfactory approximation of β.

1.4.3 Formula 2.1

Formula Z1 is a simple reformulation of the main problem of the project. It is mathematically expressed as:

$$\begin{array}{lllll} Z_1 = \min\limits_{\beta,z} & \frac{1}{2}||y - X\beta||_2^2 & & & \\ \text{s.t.} & -M_U z_i & \leq \beta_i & \leq M_U z_i & i = 1, \dots, p, \\ & z_i \in 0, 1 & & & i = 1, \dots, p, \\ & \sum\limits_{i=1}^{p} z_i \leq k & & & \end{array} \tag{8}$$

where $z \in \{0, 1\}^p$, if $z_i = 1$ the $\beta_i \leq M_U$, else $\beta_i = 0$. Concluding, $\sum_{i=1}^{p} z_i$ is an indicator of chosen features (nonzeros in β). Parameter M_U is already described in section 1.4.1.

1.4.4 Formulas 2.2 and 2.3

It is relatively easy to observe that formula 2.3 has a weaker relaxation than formula 2.3. Additionally, the formula 2.3 looks very familiar to Lasso (section 1.3) in a constrained form.

$$\begin{array}{lll} Z_2 = \min\limits_{\beta} & \frac{1}{2}||y - X\beta||_2^2 & \\ \text{s.t.} & ||\beta||_{\text{inf}} \leq & M_U \\ & ||\beta||_1 \leq & M_U k \end{array} \tag{9}$$

$$\begin{array}{lll} Z_3 = \min\limits_{\beta} & \frac{1}{2}||y - X\beta||_2^2 & \\ \text{s.t.} & ||\beta||_1 \leq & M_U k. \end{array} \tag{10}$$

1.4.5 Specially ordered sets

In the context of best subset selection, the MIO formulation based on Specially Ordered Sets can be used to mathematically represent the problem of selecting the optimal subset of variables. It involves using binary decision variables and a specially ordered set structure.
The notation $(b_i, 1 - z_i)$:SOS-1 signifies that the pair of variables $(bi, 1 - zi)$ forms a Specially

Ordered Set of Type 1 (SOS-1). A SOS-1 is a set of variables where at most one variable can take a nonzero value, and the nonzero variable must have a higher index than the one before it. This structure ensures that only one variable is selected in the subset, preventing overlapping or redundant selections.
The constraint $(1 - z_i)\beta_i = 0$ for every $i \in \{1, \ldots, p\}$ could be modeled using Specially Ordered Sets of Type 1 so that:

$$(1 - z_i)\beta_i = 0 \quad \leftrightarrow \quad (\beta_i, 1 - z_i) : \text{SOS-1} \tag{11}$$

for every $i \in \{1, \ldots, p\}$. That leads to formulations 2.4 (section 1.4.6), 2.5 (section 1.4.7) and 2.6 (section 1.4.8) of the main problem.

1.4.6 Formula 2.4

To avoid the problem of sensitivity to the choice of M_U in formula 2.1, an alternative MIO formulation using SOS was considered. The main idea behind SOS is described in section 1.4.5

$$\begin{aligned} Z_4 = \min_{\beta, z} \quad & \tfrac{1}{2}||y - X\beta||_2^2 & \\ \text{s.t.} \quad & (\beta_i, 1 - z_i) : \text{SOS-1} & i = 1, \ldots, p, \\ & z_i \in 0, 1 & i = 1, \ldots, p, \\ & \sum_{i=1}^{p} z_i \leq k. & \end{aligned} \tag{12}$$

1.4.7 Formula 2.5

Formula 2.5 is a more structured version of Fomula 2.4 (section 1.4.6). Constants M_U and M_l that appear in this formula are already described in section 1.4.1. Due to having a quadratic form and p variables, this formulation proved itself more useful in $n > p$ cases.

$$\begin{aligned} Z_5 = \min_{\beta, z} \quad & \tfrac{1}{2}\beta^T(X^TX)\beta - \langle X'y, \beta\rangle + \tfrac{1}{2}||y||_2^2 & \\ \text{s.t.} \quad & (\beta_i, 1 - z_i) : \text{SOS-1}, & i = 1, \ldots, p, \\ & z_i \in \{0, 1\} & i = 1, \ldots, p, \\ & \sum_{i=1}^{p} z_i \leq k, & \\ & -M_U z_i \leq \beta_i \leq M_U z_i, & i = 1, \ldots, p, \\ & ||\beta||_1 \leq M_l. & \end{aligned} \tag{13}$$

1.4.8 Formula 2.6

Formula 2.6 has more variables than formula 2.5 (section 1.4.7), but involves quadratic form in n of them which results in it being even more useful in $p >> n$ cases.

$$
\begin{array}{rlrl}
Z_6 = \min\limits_{\beta,z,\zeta} & \frac{1}{2}\zeta^T\zeta - \langle X'y, \beta\rangle + \frac{1}{2}||y||_2^2 & & \\
\text{s.t.} & \zeta = X\beta, & & \\
 & (\beta_i, 1-z_i) : \text{SOS-1}, \quad i = 1,\dots,p, & & \\
 & z_i \in \{0,1\}, & & i = 1,\dots,p, \\
 & \sum\limits_{i=1}^{p} z_i \leq k, & & \\
 & -M_U z_i \leq \beta_i \leq M_U z_i, & & i = 1,\dots,p, \\
 & ||\beta||_1 \leq M_l, & & \\
 & -M_U^\zeta z_i \leq \beta_i \leq M_U^\zeta z_i, & & i = 1,\dots,p, \\
 & ||\zeta||_1 \leq M_l^\zeta, & &
\end{array} \tag{14}
$$

1.4.9 Formula 2.7

This is the final formulation of the main problem. Parameters M_U, M_l, M_U^ζ and M_l^ζ are described in section 1.4.1.

$$
\begin{array}{rrl}
Z_7 = \min\limits_{\beta} & \frac{1}{2}||y - X\beta||_2^2 & \\
\text{s.t.} & ||\beta||_0 \leq & k, \\
 & ||\beta||_\infty \leq & M_U, \\
 & ||\beta||_1 \leq & M_l, \\
 & ||X\beta||_\infty \leq & M_U^\zeta, \\
 & ||X\beta||_1 \leq & M_l^\zeta.
\end{array} \tag{15}
$$

2 Contributions

In our research we were asked to implement the formulations from the Section 1.4, and inspect their behaviour. In our work, we've implemented all seven approaches, and compared them to one another for three different artificially generated datasets. Moreover we've prepared characteristic of each method for different hyperparameters values. As a bonus, we've also implemented the warmstart strategy (which is the topic for different project), and suggested method for choosing the promising M_U value. These last two methods were however left without proper evaluation as the main article doesn't provide necessary information about the choice of some of the hyperparameters.

3 Implementation

In this section, we will focus on presenting the implementation process of the methods described in the previous section. We will briefly describe the tools used in this process, their advantages and limitations, as well as we will explain how they work. The source codes are available on our GitHub repository in the form of a Jupyter notebook. Additionally, one could find there the visualizations used in this documents, and the raw results obtained after different steps.

3.1 Used tools

As the graduates of the Data Science bachelor studies, we've decided to use a solver known from a previous course, which is the IBM ILOG CPLEX solver, referred to as a Cplex later in the document. We supposed that it would suit our problems perfectly, as we were able to use both mixed integer programming methods and solve the quadratic problems.

After choosing the solver, we've decided to find a Python API for it, as we are mostly Python programmers, and we've wanted to conduct the experiments and prepare the visualizations in this programming language. Because of that, we've decided to use the official IBM package for Cplex, called docplex. In order to use it properly, and without a harsh restriction of the community edition, we had to create an IBM account, prove that we are students, download the Cplex from the official site, and install both the Cplex and docplex packages from the executable files downloaded before. The process was complex, as after installing the community version before, there were lots of issues considering the installation, which resulted in deleting both the Cplex version and starting from the beginning.
Finally, after the proper installation, we were able to write first, simple optimization problems and the tool indeed was able to work with the class of problems posed within the task of the project. The next subsection presents the implementation of the final method from the paper, and we will discuss the syntax of the solver.

3.2 Example implementation

The Listing 1 presents the docplex implementation of the final formulation of the problem described briefly in Section 1.4.9. It is the most complex method, and the other six versions are built similarly to this one. Firstly, on the input, we have to provide the matrix X with predictors, and the vector y describing the target. Despite that, the method includes 4 tunable hyperparameters and 2 auxiliary ones. The parameters M_U, M_l, M_U^ζ, and M_l^ζ are the constraint parameters, which values have a great influence on the outcomes of the method. The parameter k describes the desired number of the parameters chosen by the method, and the time_limit is an additional constraint for the maximal execution time of a single optimization process. The parameter wasn't present in the original approach, however, due to the limited computational resources we had to introduce it in order to conduct meaningful experiments.

The function starts with the creation of the model, setting its name and setting a time limit for the optimization process. The next step is creating the list of continuous variables for the optimization, which are the β parameters, corresponding to β in our problem.
The next step is the introduction of a first constraint on the norm zero of beta. It shows one of the major drawbacks of the docplex, as it is hard to introduce more complex variables or functions because the tool harshly limits the possible operations for the variables inside the model. Thus we had to create a workaround where we introduce the set of binary variables called binary_beta and in the for loop, we add an if_then rule which sets β to 0 if the corresponding binary_beta is also zero. Let's notice that such a workaround isn't a cost-free solution as we have to double the number of variables corresponding to β.
Later we define an auxiliary ksi variable which is the multiplication of X and β. Finally, we can add the constraints simply and cleanly, with the inf_pnorm, and pnorm function defined before. Let's also note that for the implementation of pnorm (Listing 2) we also create additional helper variables for the absolute values that are needed to calculate the norm.
Finally, we define the minimization task and ask the Cplex to solve it and return the outcoming model.

```python
def final_form(X, y, Mu, Ml, Mu_ksi, Ml_ksi, k, time_limit = None):
    # Getting the number of columns.
    p = X.shape[1]
    # Creating one model instance, with a name.
    m = Model(name='final_form')
    # Introducing the time limit.
    if (time_limit != None):
        m.set_time_limit(time_limit)
    # Setting the p binary variables called beta
    beta = m.continuous_var_list(p, name  = 'beta')
    # Constraint 1 with workaround
    binary_beta = m.binary_var_list(p, name = 'binary_beta')
    # We can't do a 0-norm, so we have to make a workaound by creating new
   binary variables
    for i in range(p):
        # We use an if-then statement that makes Beta a 0, when binary beta is 0
        m.add_if_then(binary_beta[i] == 0, beta[i] == 0)

    ksi = X @ beta
    # Finally we let only k binary betas to be non-zero
    const1  = m.add_constraint(sum(binary_beta)   <= k, 'const1')
    # Setting other constraints
    const2  = m.add_constraint(inf_pnorm(beta, m) <= Mu, 'const2')
    const3  = m.add_constraint(pnorm(beta, 1, m)  <= Ml, 'const3')
    const4  = m.add_constraint(inf_pnorm(ksi, m)  <= Mu_ksi, 'const4')
    const5  = m.add_constraint(pnorm(ksi, 1, m)   <= Ml_ksi, 'const5')

    m.minimize(norm2_2((y - X @ beta))/2)
    m.solve()
    return(m)
```

Code Listing 1: Implementation of the final forumlation of the problem presented in the paper.

```python
def pnorm(x, p, model):
    if p == 1:
        x = [model.abs(i) for i in x] # it adds more variables
        x = sum(x)
    return x
```

Code Listing 2: Implementation of the pnrom function.

3.3 Issues with implementation

Unfortunately, as mentioned before docplex is extremely limited in comparison to the classical Cplex Studio, and we've encountered lots of issues considering the implementation.

- Firstly, it lacks the possibility to compute some of the most basic mathematical functions such as the root of a variable, the division of them, or calculating the correlations, which makes it hard or even impossible to implement some methods used in a theoretical way,
- There are no predefined functions for calculating a bit more complex functions such as p-norm, or 0-norm,
- Although the documentation is well written, it lacks any official examples of how to use some methods or parameters, and the answers on forums such as stack-overflow are often incorrect,
- Even though, sometimes there exists a docplex implementation of basic functions, such as absolute value, it creates a new variable for each absolute value created, which makes the optimization tasks with a huge number of variables. This issue can be extremely problematic for the users that want to work with a community edition, as we easily use up the limit of 1000 variables.

4 Experiments

In this section we will introduce the experiments conducted with the usage of implemented methods. It contains the description of generated datasets, experiments pipelines, and the description of the outcomes.

4.1 Datasets

The datasets used for the experiments were generated according to the formulas proposed in the main article. Unfortunately, we weren't able to implement all artificially generated datasets, as some of them were poorly described, and contained parameters whose values weren't given at any point of the paper. This way we've omitted the first experimental dataset of the article, but we've managed to implement the other three. To fulfill the condition p >> n, we've decided to generate small datasets with 100 columns (p / features), and only 30 observations, which made the task of proper feature selection very demanding. The increase of the sizes was impossible, as we had access to the limited computational resources, and wanted to conduct a wide study of hyperparameters.

4.1.1 Datasets description

Equal coefficients: For the first task we've generated the dataset, where predictors come from o normal multivariate distribution, based on a covariance matrix X having 1s on the diagonal and 0s everywhere else. The values were later normalized with the l2 norm. The target value was calculated based on the 5 first features, which had the same weight equal to 1, whereas the rest had weights equal to 0. To the y values we've also added a noise coming from the normal distribution with the mean 0 and standard deviation equal to $\frac{Var(X\beta)}{\text{SNR}}$, where SNR stands for Signal-To-Noise ration and is set to 7, and $\beta = [1, 1, 1, 1, 0, ..., 0]$.

It was the most simple task, where all the true values had the same weights being much bigger than irrelevant features, thus their detection was plain and simple for most of the algorithms, and they didn't have issues with selecting proper columns as relevant.

Growing coefficients: For the second task the matrix of predictors X was generated in the same way as before, but we've assigned different coefficients for relevant features. The relevant features were the first ten, and the i-th column had the weight equal to $\frac{1}{2} + 9.5(\frac{i}{10})$. It means that their impacts were growing from 1st feature till the 10th. The y values were generated similarly as before but with β equal to the coefficients described here.

This method was a challenge for the proposed optimization algorithms, as with unequal weights they were focusing mostly on features 5-10 which had bigger weights, and had major issues with selecting the features with smaller weights as significant. The most interesting behavior happened for the SOS2 formulation 1.4.7, where the helper variables z correctly selected all 10 features, but non-zero β values were assigned only for the last three of them. Further details will be described later.

Negative coefficients: For the last task the matrix of predictors X was generated in the same way as before, but we've assigned different coefficients for relevant features. The relevant features were the first six, and their values were equal to [-10, -6, -2, 2, 6, 10]. This task was extremely difficult as proper features provide opposite impacts to one another. The y values were generated similarly as before but with β equal to the coefficients described here.

4.2 Experiments main pipeline

As our study we've wanted to measure the impact of different values of M_U, M_l, M_U^ζ, and M_l^ζ parameters for the methods that use them. To provide such characteristics, we've decided to compute the performance of all seven methods for a grid searched values of the aforementioned parameters equal to $[0.01, 0.1, 1, 10]$. For each optimization problem evaluation, we saved the final parameter values, with z and β being the most important ones, the objective function value, corresponding hyperparameters used for the evaluation, and the time spent on the computations of the particular method.

At this point we've faced a major issue considering the time of calculations. For some sets of hyperparameters (for example when only one of the M parameters was different than the others), the optimization process could last for hours and never converge. For this reason, we've introduced the aforementioned timeout parameter for each method and limited the maximal calculation time to one minute.

The raw results were later evaluated with a few approaches. The basic one considered the analysis of the objective value returned by the solver. The other options followed our intuition, that the most important part of the implementation is the feature selection. Because of that, as our results, we've treated the non-zero β or z coefficients chosen by the methods. For them, we've calculated the Sensitivity, Specificity, and Balanced Accuracy, where the last metric is the most important one. We've decided to use these metrics, as we can treat our task as the imbalanced binary classification, and those values are perfect for describing such a problem. Additionally, we've decided to calculate separate scores for parameters β and z, and see if there is a big difference between the predictions made by those two.

4.2.1 Modification for SOS3 and Final

As the last two methods (SOS3, and the Final method) are the most prominent solutions, we've decided to take a closer look at their behavior depending on different parameters. In order to do it, we've taken the parameters of best solutions and conducted research on all 4 parameters M_U, M_l, M_U^ζ, and M_l^ζ. For every parameter we've calculated new solutions with fixed 3 out of 4 parameters, and the values for the left out one were: $[0.01, 0.05, 0.1, 0.5, 1, 5, 10]$. For example, we froze parameters M_l, M_U^ζ, and M_l^ζ at 0.1 values, and run methods SOS3 and Final with M_U equal to $[0.01, 0.05, 0.1, 0.5, 1, 5, 10]$. This experiment let us analyze the impact of each hyperparameter on the performance of the method. The results were provided in the same way as for other methods.

5 Outcomes

The last section provides the outcomes of our analysis focusing on the hyperparameter influence on the outcomes of the methods. As suspected before, the best results were obtained for the first experiment, the moderate ones for the second, and the last task proved to be the most difficult one. We will present the results for the groups of methods, depending on the number of hyperparameters used (1, 2, or 4, we will omit the SOS1 method as it was a non-parametric one), and we will comment on the results for each dataset separately.

5.1 Z1, Z2, Z3 Formulations

Firstly, let's describe the results for the Z-methods which had only one hyperparameter to tune.

5.1.1 Dataset 1

The Figure 1 presents the performance of Z1, Z2, and Z3 methods (Sections 1.4.3, 1.4.4) depending on the value of parameter M_u for a first dataset. The plots on the left present balanced accuracy, whereas the ones on the right describe the objective value returned by the algorithm. The dots sizes describe the computational time for each evaluation, which was much faster for algorithms Z2 and Z3. For the first case algorithm Z1, the results are the same for both β and z parameters which results in a single line on the plot. In general, the best results are able to reach optimal or sub-optimal solutions.

As we can see, higher M_U values resulted in better results for the Z1 algorithm and it was able to achieve perfect scores for this method.
The second method is more interesting, as it was unable to achieve the perfect balanced accuracy, even though it reached the optimal objective value. The interesting part is that for the $M_U =$ 1the method got higher balanced accuracy than for the $M_U = 10$, even though the objective value was optimal in both cases.
The last method's results lead to the same conclusion as the Z2, although we can see that in terms of balanced accuracy, it was less stable than the previous one.

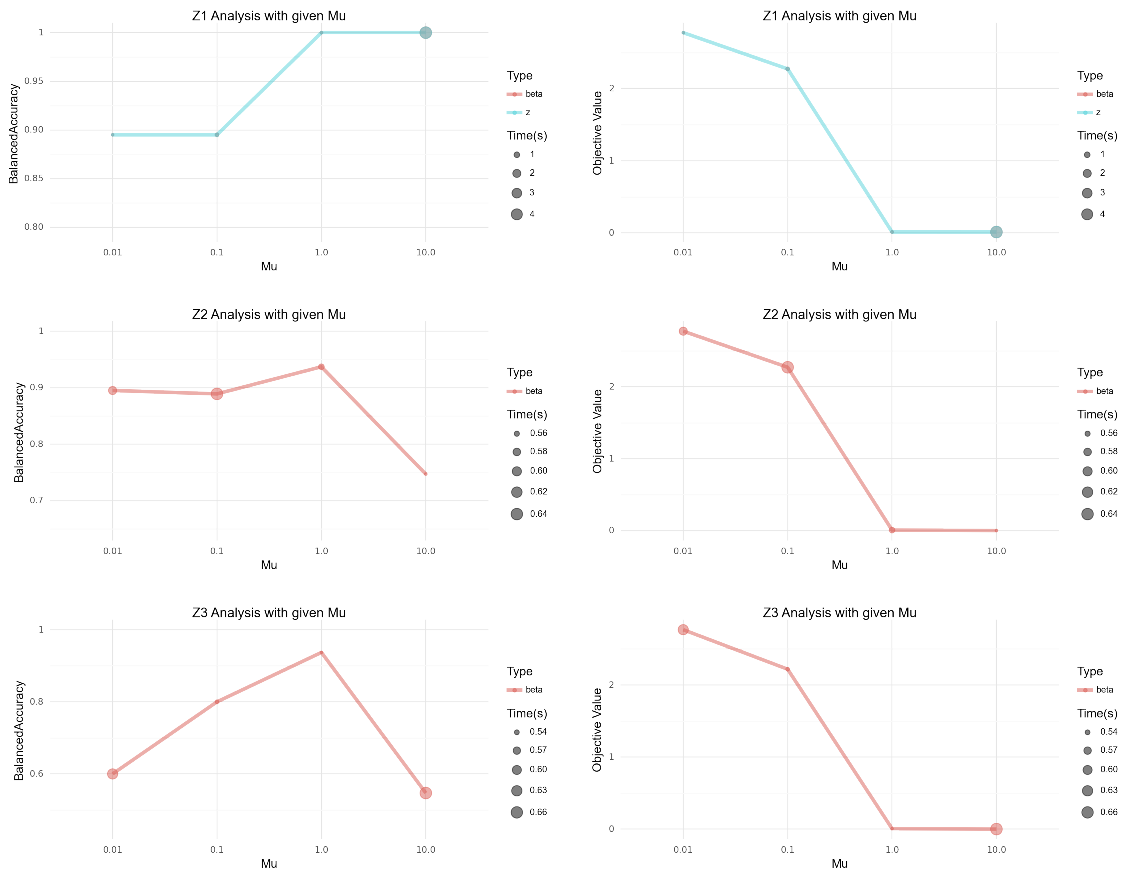

Figure 1: The M_U parameter analysis for Z1, Z2, and Z3 methods for the task 1.

5.1.2 Dataset 2

The Figure 2 presents the performance of Z1, Z2, and Z3 methods depending on the value of parameter M_u for a second dataset. The plots on the left present balanced accuracy, whereas the ones on the right describe the objective value returned by the algorithm. The dots sizes describe the computational time for each evaluation, which was a bit faster for algorithms Z2 and Z3. For the first case algorithm Z1, the results are the same for both β and z parameters which results in a single line on the plot. We can easily see that this task is harder than the previous one, and the achieved outcomes are worse.

As we can see, different M_u values don't change a lot for the Z1 algorithm in case of balanced accuracy and its performance was similar for all values, even though the objective value was diminishing, the bigger the M_U was.
The second and third methods were more interesting and performed much better than the most basic ones. The loss functions looked similar and ended up being close to 0, whereas the balanced accuracy for both methods was getting higher, the bigger the M_U parameter was. Finally, the Z2 method achieved a balanced accuracy for selected variables over 0.8 which is a decent score.

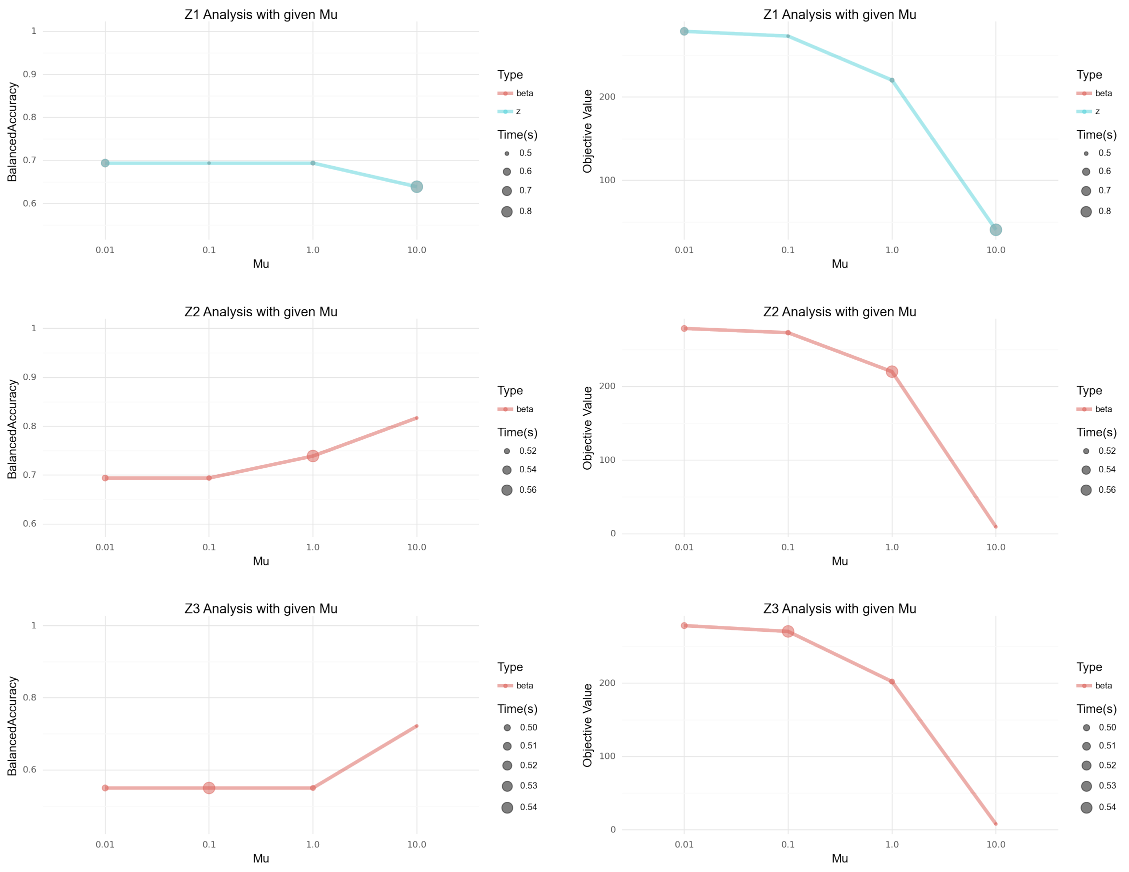

Figure 2: The M_u parameter analysis for Z1, Z2, and Z3 methods for the task 2.

5.1.3 Dataset 3

The Figure 3 presents the performance of Z1, Z2, and Z3 methods depending on the value of parameter M_u for a second dataset. The plots on the left present balanced accuracy, whereas the ones on the right describe the objective value returned by the algorithm. The dots sizes describe the computational time for each evaluation, which was much faster for algorithms Z2 and Z3. For the first case algorithm Z1, the results are the same for both β and z parameters which results in a single line on the plot. The last task is definitely the hardest one, as achieved balanced accuracy values are below 0.6, which shows us that our solutions behave poorly in this case.

As we can see, different M_U values don't change a lot for the Z1 algorithm in case of balanced accuracy and its performance was similar for all values, even though the objective value was diminishing, the bigger the M_U was.
The second and third methods are getting worse in terms of balanced accuracy, the higher the M_u value is, even though its objective values are getting better. It means that although we omit some important parameters, the method manages to assign good β coefficients which results in high predictive power.

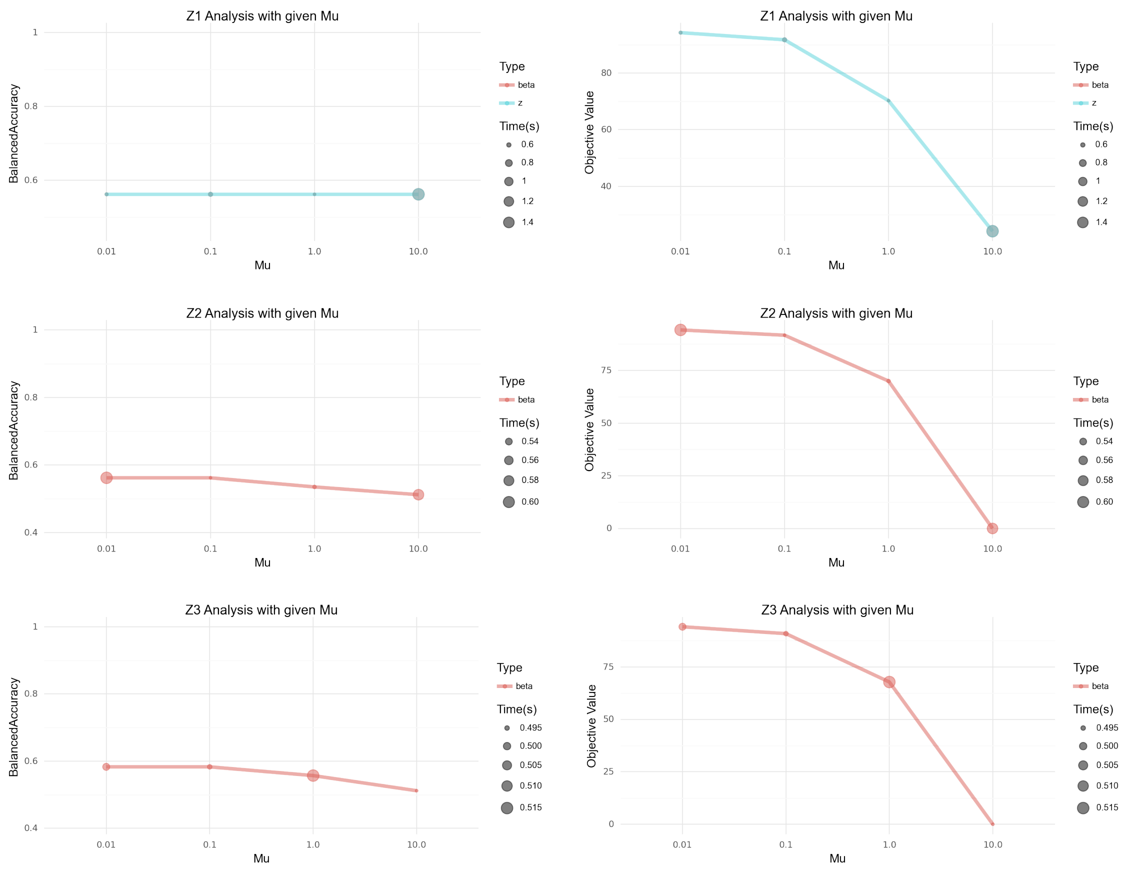

Figure 3: The M_u parameter analysis for Z1, Z2, and Z3 methods for the task 3.

5.2 SOS2 Formulation

Now, let's analyze the results obtained by the SOS2 formulation (Section 1.4.7). This time we had to tune 2 parameters M_U and M_l, thus the results are presented in the form of 4 heatmaps, depending on the parameters' values. The first one shows us the balanced accuracy for the helper parameter z, whereas the second one is for the β. The third plot shows the distribution of the objective value, and the last one presents how long each execution lasted.

5.2.1 Dataset 1

As we can see from Figure 4, the SOS2 formulation proved to be extremely efficient for the first task, as it achieved both perfect balanced accuracy scores, as well as almost 0 objective values. Moreover, thanks to this analysis we can clearly witness that for some parameter sets, the z variables correctly indicated important variables, even though the β coefficients were equal to 0. Moreover, we can witness that the bigger, both M_u and M_l values were, the better the obtained results were. Another interesting observation is that setting M_U to 0.01 resulted in the worst results, and further tuning of the M_l parameter didn't results in any improvement. Based on such observations we can say that the constraint with M_U impacts the whole optimization process more than the one with M_l. Finally, we can also find out, that the method was incredibly fast, and the bigger both parameters were, the longest the computation took.

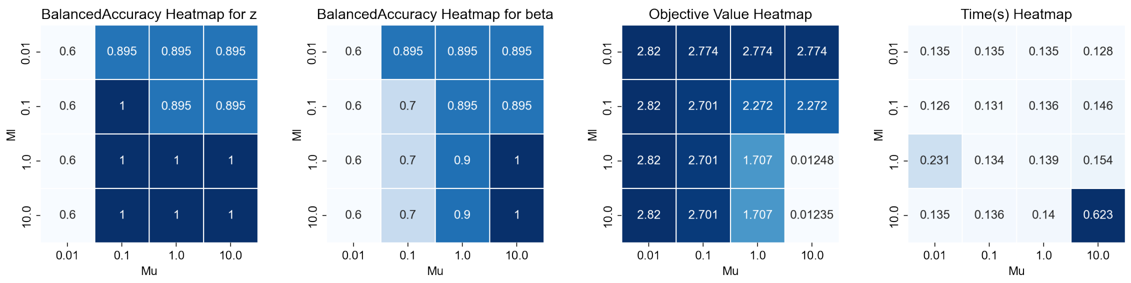

Figure 4: The M_U and M_l parameters analysis for SOS2 method for the task 1.

5.2.2 Dataset 2

The Figure 5 shows us that the second task was indeed harder than the previous one, as obtained results were far worse in terms of balanced accuracy. The method proved to achieve worse results than the Z1, Z2, and Z3 formulations, as the best balanced accuracy value for β (0.7) was lower than the one for Z3 (around 0.8), and the objective value was also much higher (150 instead of around 0). We could probably achieve better results if we considered even higher values for M_u and M_l, as the trend from the previous tasks holds in this one, which means that the bigger those values the better the results were.

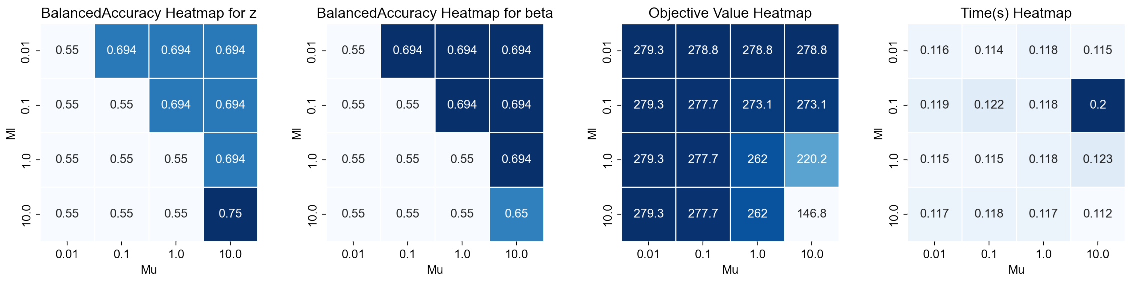

Figure 5: The M_U and M_ll parameters analysis for SOS2 method for the task 2.

5.2.3 Dataset 3

The Figure 6 shows us that the last task was the hardest one, and results obtained by this method were similar in terms of balanced accuracy to the ones obtained before, by the Z formulations. According to the raw data, only one feature out of 6 was chosen properly, thus the Sensitivity of the method was around 0.17 when the Specificity was around 1. Perhaps, a bigger search space would enable us to end up with better results, as increasing the M_U and M_l values might lead to better results, according to the third plot.

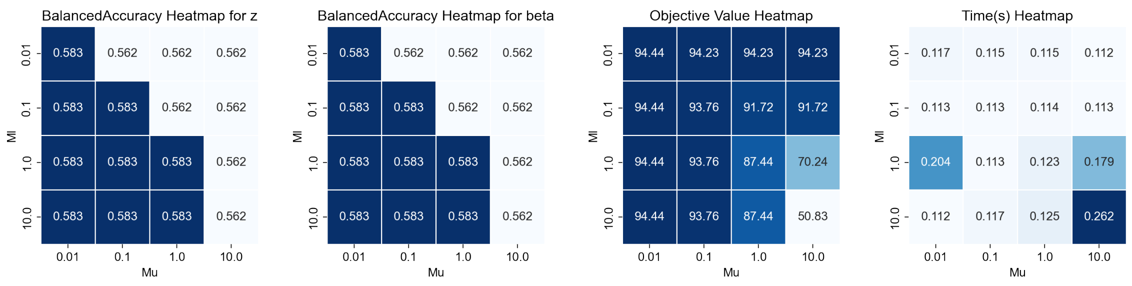

Figure 6: The M_u and M_l parameters analysis for SOS2 method for the task 3.

5.3 SOS3 and Final Formulations

Finally we can analyze the impact of hyperparameters for the most advanced SOS3 and Final formulations (Sections 1.4.8, and 1.4.9). This time we had a 4-dimensional search space, and as the methods were the slowest ones considered in our research, we had to limit the search space according to this method, and that's why we've decided to provide only 4 values for all experiments in our study. In this case, we weren't able to present the results of 4-dimensional space clearly on any plot, thus we've decided to pick the hyperparameters for both methods that yielded the best results from our grid search, and then commit a study where we freeze 3 out of 4 parameters, and create new models by changing only the last one.
Because of that, we've ended up having 4 lines on each plot, representing the tuning of different parameters and drawing its characteristic curve. As these were the most important methods, we've decided to consider 7 values instead of 4, being $0.01, 0.05, 0.1, 0.5, 1, 5, 10]$. The plots in the left column represent the balanced accuracy scores, whereas the plots on the right show the objective value of the model. The rows describe different methods, and the exact hyperparameters on which the tunings are based are described in the plot titles.

5.3.1 Dataset 1

Accordingly to the previous method, neither of these two had any problems in obtaining perfect results. Moreover, from Figure 7 we can clearly see that the higher the parameters were the better both balanced accuracy and objective value were. The disturbing fact however is the execution time, which was between 20 to 60 seconds, even though the first task was very simple. Let's remember that we've limited the optimization time for each model to 1 minute, which means that very often this limit was reached. This time, the results were good, although in some cases, this constraint could have a negative impact on the performance.

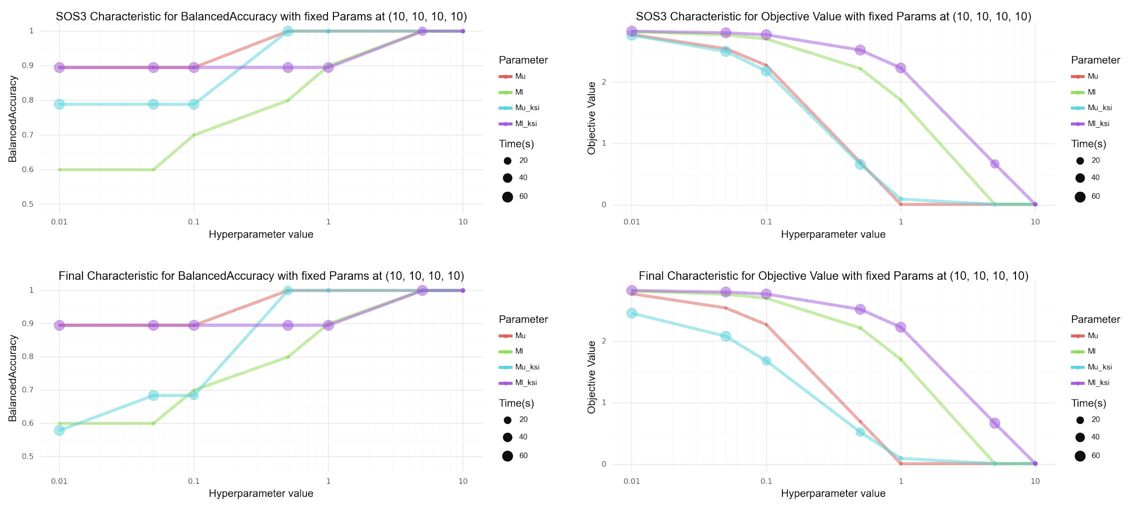

Figure 7: The M_U, M_l, M_U^ζ, and M_l^ζ parameters analysis for SOS3 and Final methods for the task 1.

5.3.2 Dataset 2

The first thing we can notice from Figure 8, is faster execution times than for the previous task. Another interesting observation is the analysis of the parameter Mu curves, which was getting worse results for balanced accuracy when its value was growing, whereas for all parameters the objective value was diminishing. It's another case when the outcomes could benefit from expanding our hyperparameter search space by bigger values. We can also notice, that in this case, the results were the best of all in terms of balanced accuracy.

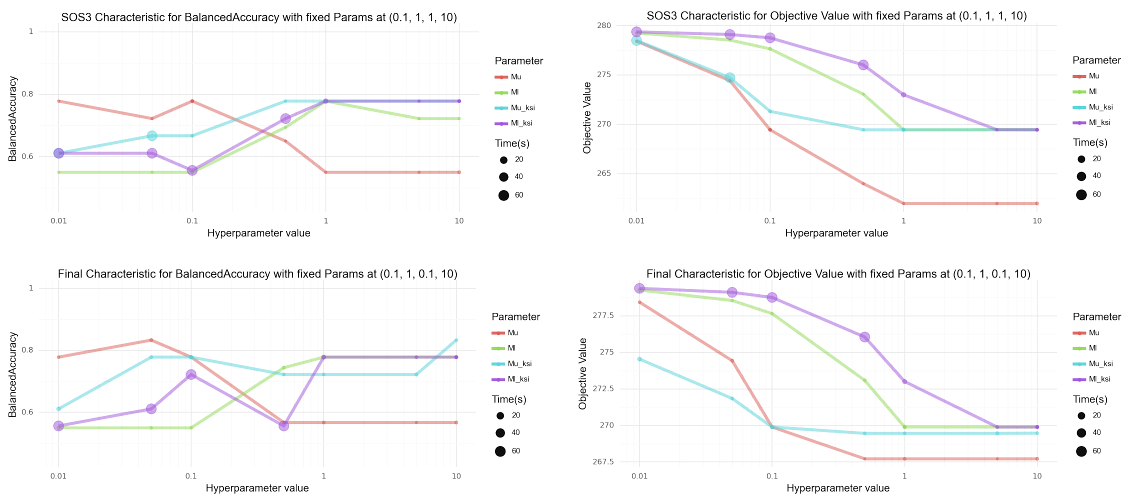

Figure 8: The M_u, M_l, M_u^ζ, and M_l^ζ parameters analysis for SOS3 and Final methods for the task 2.

5.3.3 Dataset 3

The last analysis presented in Figure 9 is extremely interesting, as this is the first time when we could achieve a balanced accuracy slightly higher than 0.6, which is the best result of all. Moreover, these results prove that this task is extremely hard, as even the tuning of 4 different hyperparameters didn't results in big changes in the results.

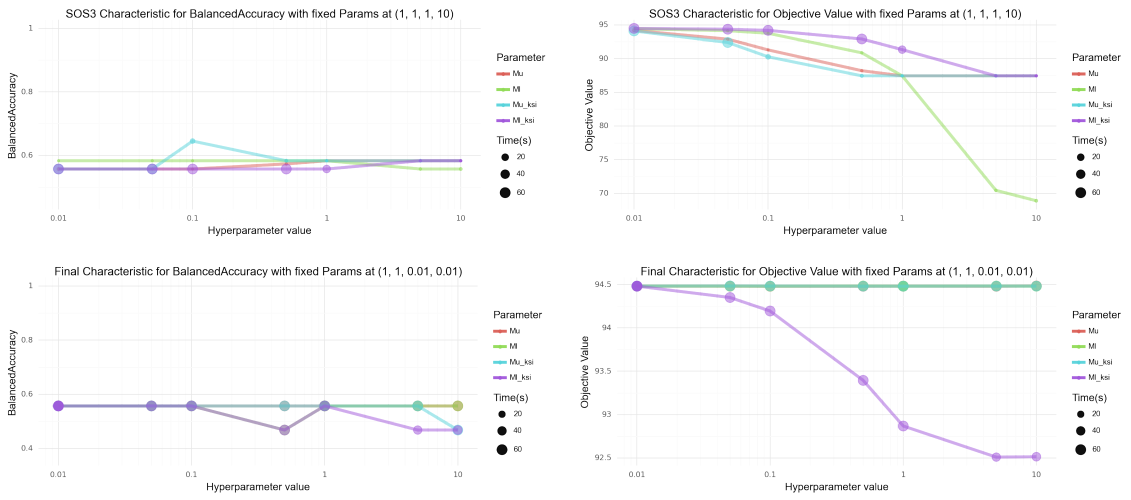

Figure 9: The M_u, M_l, M_u^ζ, and M_l^ζ parameters analysis for SOS3 and Final methods for the task 3.

6 Remarks to the article

- The authors often provided algorithms and formulas with certain hyperparameters without specifying what should be their value nor how to calculate them (UB value from section 2.3.2, τ in section 2.3.3, L and ϵ in Algorithm 1),
- Results achieved by the authors are impossible to reproduce due to a lack of information about used parameters,
- Synthetic dataset presented in as Example 1 in section 5.1 in the article was unable to reproduce due to lack of explanation of used symbols,
- Lack of information about initialization of β_1 in section 3.1 in Algorithm 1.

7 Further works

Despite our best intentions and high devotion to the project, we are aware of the limitations of our research, and we propose some aspects where we would seek improvement in further works:

- First and foremost, as stated multiple times in the section describing the results of our research, our study would greatly benefit from expanding the hyperparameters search space, especially, by including higher parameter values. It could be than in two separate ways: generally add more values for all experiments, or at least do it for the formulations Z1, Z2, Z3, and SOS2, as these methods were working very fast.
- We could also train the linear regression models on the proposed β coefficients and see compare their performance instead of comparing the objective values.
- We could also use bigger datasets with more observations, and/or more hyperparameters, to analyze the behavior of the methods in different environments.
- Another beneficial area would be the analysis of the performance for real-life datasets, such as leukemia. We didn't consider this task in our study, as it requires a different evaluation approach, as we don't know which parameters are good predictors.
- With more time and computational resources, we could also compare this feature selection method to other well-established solutions, such as BORUTA, ridge, and lasso regressions, or even the methods based on mutual information metrics. Finally, we could evaluate the impact of the warm-starts and how the method of choosing the correct upper bound on Mu, and Ml parameters benefits the model.

8 Thoughts and conclusions

Let's briefly sum up our research. During the project we've implemented all problem formulations, on our own with the docplex Python API for Cplex solver, without using any existing solutions. Additionally we've implemented warmstarts and hyperparameter choosing methods, also described in the main papers. Furthermore, we've conducted a wide study of the different problem formulations behaviour, when different sets of hyperparameters were provided, and described the outcomes in this document.

Generally, the project was fairly interesting, as we could witness a different approach for the feature selection method, unexplored during the course of data science studies. Moreover we had a chance to revisit our skills for solving the optimization tasks with the Cplex solver. Using the Python API in the form of docplex package was at first interesting and promising, but it quickly turned into a huge disappointment as it proved itself to be a tool unprepared for a wide usage. Because of mediocre documentation, lack of examples and very vague error messages we wasted time blindly debugging the code. Moreover, some things are basically impossible to implement in this framework, but we still had to waste hours for trying to overcome such issues.

Additionally, the problem of linear regression is a rather common one. It was interesting to look at the problem of feature selection for this task from a more mathematically-optimisational way, but on the other hand it left us with a feeling that we're solving an already solved problem, and final results were not satisfactory enough.

Finally, the main paper was often too vague, poorly written or even lacked parameters specifications, which made a task of implementing these solutions very hard, and sometimes even impossible. On the other hand, the paper yielded some positives. In order to implement the methods presented in the article we had to spend lots of time for studying it, which lead to better understanding of optimization in data analysis and deepening our mathematical and machine learning knowledge.

References

[Bertsimas et al., 2015] Bertsimas, D., King, A., and Mazumder, R. (2015). Best subset selection via a modern optimization lens.

[Hoerl and Kennard, 2000] Hoerl, A. E. and Kennard, R. W. (2000). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 42(1):80–86.

[Kursa and Rudnicki, 2010] Kursa, M. B. and Rudnicki, W. R. (2010). Feature Selection with the Boruta Package. *Journal of Statistical Software*, 36(11):1–13.

[Sulaiman and Labadin, 2015] Sulaiman, M. A. and Labadin, J. (2015). Feature selection based on mutual information. pages 1–6.

[Tibshirani, 2018] Tibshirani, R. (2018). Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

List of Figures