LEARNING TO RELAX NONCONVEX QUADRATICALLY CONSTRAINED QUADRATIC PROGRAMS

by
M. BUKET ÖZEN

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Master of Science

Sabancı University
July 2024

LEARNING TO RELAX NONCONVEX QUADRATICALLY CONSTRAINED QUADRATIC PROGRAMS

Approved by:

Assoc. Prof. Burak Kocuk ....
(Thesis Supervisor)

Prof. İsmail Kuban Altınel ....

Asst. Prof. Sinan Yıldırım ....

Date of Approval: July 19, 2024

ABSTRACT

LEARNING TO RELAX NONCONVEX QUADRATICALLY CONSTRAINED QUADRATIC PROGRAMS

M. BUKET ÖZEN

INDUSTRIAL ENGINEERING M.S. THESIS, JULY 2024

Thesis Supervisor: Dr. Burak Kocuk

Keywords: quadratically constrained quadratic program, linear programming relaxations, semidefinite programming relaxations, global optimization, machine learning, classification, regression

Quadratically constrained quadratic programs (QCQPs) are commonly encountered in diverse disciplines like operations research, machine learning, power systems, and portfolio optimization. The solution of these non-convex problems is difficult since they are characterized by their NP-hard nature. Conventional methods employ either Semidefinite Programming (SDP) or Linear Programming (LP) relaxations; each of these methods is highly effective for certain subsets of problems and exhibits poor performance for others. However, there is a lack of comprehensive understanding. This thesis seeks to create a relaxation selection procedure for QCQPs that takes into account the structure of the problem. It intends to determine if an SDP- or LP-based strategy is more beneficial based on the instance structure.

We explore the spectral properties and sparsity patterns of data matrices to unravel the structural properties of a QCQP instance. Our study is based on the generation of QCQP samples, along with an exploratory analysis of the dataset, and applying machine learning to classify the obtained instances by the most favorable relaxation technique.

The main contribution of this work is to develop machine learning models that can accurately predict ex-ante whether an SDP or LP relaxation will yield a tighter bound on new QCQP instances. These models are trained on features created from the spectral properties and sparsity patterns of the data matrices. This predictive capability increases the efficiency of solving non-convex QCQPs by guiding the selection of the most suitable relaxation method.

ÖZET

DIŞBÜKEY OLMAYAN KUADRATİK KISITLI KUADRATİK PROGRAMLARI GEVŞETMEYİ ÖĞRENMEK

M. BUKET ÖZEN

ENDÜSTRİ MÜHENDİSLİĞİ YÜKSEK LİSANS TEZİ, TEMMUZ 2024

Tez Danışmanı: Dr. Burak Kocuk

Anahtar Kelimeler: kuadratik kısıtlı kuadratik programlar, doğrusal programlama gevşetmesi, yarı tanımlı programlama gevşetmesi, makine öğrenimi, sınıflandırma, regresyon

Kuadratik kısıtlı kuadratik programlar (QCQP), yöneylem araştırması, makine öğrenimi, güç sistemleri ve portföy optimizasyonu gibi çeşitli alanlarda sıkça karşımıza çıkar. Bu dışbükey olmayan problemlerin çözülmesi, NP-zor doğaları nedeniyle oldukça zordur. Geleneksel yaklaşımlar ya Yarı Tanımlı Programlama (SDP) ya da Doğrusal Programlama (LP) gevşetmeleri kullanır; her biri belirli problem türleri için güçlü yönler sergiler ancak çeşitli örnekler arasında kapsamlı bir anlayıştan yoksundur. Bu çalışmada, dışbükey olmayan QCQP'ler için problem örneğinin yapısını göz önünde bulundurarak SDP veya LP tabanlı yaklaşımlardan hangisinin daha başarılı olacağını tahmin eden *yapıya duyarlı* bir gevşetme seçim yöntemi üretmeyi amaçlıyoruz.

QCQP örneklerindeki veri matrislerinin negatif özdeğerlerinin sayısı, özdeğerlerinin büyüklükleri, bu matrislerin seyreklik örüntüsü gibi bir takım özniteliklerini inceleyerek yapısal özelliklerini ortaya çıkarıyoruz. Araştırmamız, QCQP örnekleri oluşturmayı, keşifsel veri analizleri yapmayı ve bu örnekleri SDP-lehine veya LP-lehine olarak sınıflandırmak için makine öğrenimi tekniklerini uygulamayı içerir.

Bu tezin katkıları arasında, yeni QCQP örnekleri için SDP veya LP gevşetmesinin daha güçlü bir sınır sağlayacağını doğru bir şekilde tahmin eden makine öğrenimi modellerinin geliştirilmesi bulunmaktadır. Bu modeller, veri matrislerinin spektral özellikleri ve seyreklik örüntülerinden türetilen özniteliklerle eğitilmiştir. Bu tahmin yeteneği, en uygun gevşetme yönteminin seçimini yönlendirerek dışbükey olmayan QCQP'lerin çözüm verimliliğini artırmaktadır.

ACKNOWLEDGEMENTS

First and foremost, I would like to start by sincerely thanking Dr. Burak Kocuk. Your exceptional knowledge and steadfast support have become the foundation of this thesis. This journey has been gratifying and enlightening because of your wise counsel, patience, and support. I am profoundly grateful of the opportunity to work under your supervision as well as the countless ways you have contributed to both my academic and personal growth. I firmly believe that I am better equipped for the next phase of my career thanks to the time we have shared together.

I extend my gratitude to my family and friends for their continual support throughout my academic journey. To my family: I am deeply grateful to my parents for their unconditional love. Mom and Dad, your belief in me has always been my greatest source of strength and it significantly shaped the person I am today. The invaluable life lessons you've imparted are as significant as anything I learned during my time at Sabancı University. To Bahar and Simay: your presence has consistently made my days brighter, even in the darkest moments. Finally, and most importantly, I wish to express my heartfelt gratitude to my late grandmother, whose wisdom and love continue to inspire me. Her memory has been a guiding light throughout my academic journey.

TABLE OF CONTENTS

LIST OF TABLES .... **xi**

LIST OF FIGURES .... **xiv**

1. INTRODUCTION **1**

2. INSTANCE GENERATION **7**

2.1. Matrices with Different Sparsity Patterns 8

2.1.1. Hollow Matrices 8

2.1.2. Bipartite Graph Matrices 8

2.1.3. Random Spanning Tree Matrices 9

2.1.4. Planar Graph Matrices 9

2.1.5. Chordal Graph Matrices 10

2.1.6. Conversion of Adjacency to Distance Matrices 11

2.1.7. Diagonal Matrices 12

2.2. Random Matrices 14

2.2.1. Random Symmetric Matrix 14

2.2.2. Random Matrices with Specified Eigenvalues 14

2.2.3. Random Symmetric Positive Definite Matrix 15

2.3. QCQPs without Finite Variable Bounds 16

3. RESEARCH METHODOLOGY **18**

3.1. Optimization Modeling Languages and Related Solvers 18

3.2. Exploratory Data Analysis 19

3.3. Learning Experiments 23

3.3.1. Feature Design 23

3.3.1.1. Fully Dimension-Dependent Setup 24

3.3.1.2. Semi Dimension-Dependent Setup 25

3.3.1.3. Dimension-Independent Setup 26

3.3.2. Data Labeling Workflow 32

3.3.3. Training and Testing 33

4. RESULTS **34**
4.1. Classification Model Performance 36
4.2. Regression Model Performance 40

5. CONCLUSION **45**

BIBLIOGRAPHY **47**

APPENDIX A **50**

LIST OF TABLES

Table 3.1. Supported Problem Types by Various Optimization Software Packages. 19

Table 3.2. Distribution of Data: LP-favoring vs. SDP-favoring for Different Number of Decision Variables and Constraints. 19

Table 3.3. Percentages of Different Metrics for A_1 and A_2. 20

Table 3.4. Comparison of LP and SDP Relaxations for Convex and Reverse Convex Problems. 22

Table 3.5. Description of Fully Dimension-Dependent Setup. 25

Table 3.6. Description of Semi Dimension-Dependent Setup. 26

Table 3.7. Description of Dimension-Independent Setup. 29

Table 3.8. Example features with $n = 3, m = 2$. 31

Table 4.1. List of Experiments. 35

Table 4.2. Best Classification Metrics for Different Experiments (LP vs. SDP). 36

Table 4.3. Best Classification Metrics for Different Experiments (LP vs. SDP'). 40

Table 4.4. Best Regression Metrics for Different Experiments (LP vs. SDP). 41

Table 4.5. Best Regression Metrics for Different Experiments (LP vs. SDP'). 43

Table A.1. Classification Metrics for D-5-1 (LP vs. SDP) 50

Table A.2. Classification Metrics for D-5-2 (LP vs. SDP) 50

Table A.3. Classification Metrics for D-10-1 (LP vs. SDP) 50

Table A.4. Classification Metrics for D-10-2 (LP vs. SDP) 51

Table A.5. Classification Metrics for N-5-1 (LP vs. SDP) 51

Table A.6. Classification Metrics for N-5-2 (LP vs. SDP) 51

Table A.7. Classification Metrics for N-5-3 (LP vs. SDP) 51

Table A.8. Classification Metrics for N-5-10 (LP vs. SDP) 52

Table A.9. Classification Metrics for N-10-1 (LP vs. SDP) 52

Table A.10.Classification Metrics for N-10-2 (LP vs. SDP) 52

Table A.11.Classification Metrics for N-5,10-1 (LP vs. SDP) 52

Table A.12.Classification Metrics for N-5,10-2 (LP vs. SDP) 53
Table A.13.Classification Metrics for N-5,10-2' (LP vs. SDP) 53
Table A.14.Classification Metrics for I-5-2 (LP vs. SDP) 53
Table A.15.Classification Metrics for I-5-10 (LP vs. SDP) 53
Table A.16.Classification Metrics for I-10-2 (LP vs. SDP) 54
Table A.17.Classification Metrics for I-10-2' (LP vs. SDP) 54
Table A.18.Classification Metrics for D-5-1 (LP vs. SDP') 54
Table A.19.Classification Metrics for D-5-2 (LP vs. SDP') 55
Table A.20.Classification Metrics for D-10-1 (LP vs. SDP') 55
Table A.21.Classification Metrics for D-10-2 (LP vs. SDP') 55
Table A.22.Classification Metrics for N-5-1 (LP vs. SDP') 55
Table A.23.Classification Metrics for N52 (LP vs. SDP') 56
Table A.24.Classification Metrics for N-5-3 (LP vs. SDP') 56
Table A.25.Classification Metrics for N-5-10 (LP vs. SDP') 56
Table A.26.Classification Metrics for N-10-1 (LP vs. SDP') 56
Table A.27.Classification Metrics for N-10-2 (LP vs. SDP') 57
Table A.28.Classification Metrics for N-5,10-1 (LP vs. SDP') 57
Table A.29.Classification Metrics for N-5,10-2 (LP vs. SDP') 57
Table A.30.Classification Metrics for N-5,10-2' (LP vs. SDP') 57
Table A.31.Classification Metrics for I-5-2 (LP vs.SDP') 58
Table A.32.Classification Metrics for I-5-10 (LP vs. SDP') 58
Table A.33.Classification Metrics for I-10-2 (LP vs. SDP') 58
Table A.34.Classification Metrics for I-10-2' (LP vs. SDP') 58
Table A.35.Regression Metrics for D-5-1 (LP vs. SDP) 59
Table A.36.Regression Metrics for D-5-2 (LP vs. SDP) 59
Table A.37.Regression Metrics for D-10-1 (LP vs. SDP) 59
Table A.38.Regression Metrics D-10-2 (LP vs. SDP) 59
Table A.39.Regression Metrics for N-5-1 (LP vs. SDP) 59
Table A.40.Regression Metrics for N-5-2 (LP vs. SDP) 60
Table A.41.Regression Metrics for N-5-3 (LP vs. SDP) 60
Table A.42.Regression Metrics for N-5-10 (LP vs. SDP) 60
Table A.43.Regression Metrics for N-10-1 (LP vs. SDP) 60
Table A.44.Regression Metrics for N-10-2 (LP vs. SDP) 60
Table A.45.Regression Metrics for N-5,10-1 (LP vs. SDP) 61
Table A.46.Regression Metrics for N-5,10-2 (LP vs. SDP) 61
Table A.47.Regression Metrics for N-5,10-2' (LP vs. SDP) 61
Table A.48.Regression Metrics for I-5-2 (LP vs. SDP) 61
Table A.49.Regression Metrics for I-5-10 (LP vs. SDP) 61
Table A.50.Regression Metrics for I-10-2 (LP vs. SDP) 62

Table A.51.Regression Metrics for I-10-2' (LP vs. SDP) 62
Table A.52.Regression Metrics for D-5-1 (LP vs. SDP') 62
Table A.53.Regression Metrics for D-5-2 (LP vs. SDP') 62
Table A.54.Regression Metrics for D-10-1 (LP vs. SDP') 63
Table A.55.Regression Metrics for D-10-2 (LP vs. SDP') 63
Table A.56.Regression Metrics for N-5-1 (LP vs. SDP') 63
Table A.57.Regression Metrics for N-5-2 (LP vs. SDP') 63
Table A.58.Regression Metrics for N-5-3 (LP vs. SDP') 63
Table A.59.Regression Metrics for N-5-10 (LP vs. SDP') 64
Table A.60.Regression Metrics for N-10-1 (LP vs. SDP') 64
Table A.61.Regression Metrics for N-10-2 (LP vs. SDP') 64
Table A.62.Regression Metrics for N-5,10-1 (LP vs. SDP') 64
Table A.63.Regression Metrics for N-5,10-2 (LP vs. SDP') 64
Table A.64.Regression Metrics for N-5,10-2' (LP vs. SDP') 65
Table A.65.Regression Metrics for I-5-2 (LP vs. SDP') 65
Table A.66.Regression Metrics for I-5-10 (LP vs. SDP') 65
Table A.67.Regression Metrics for I-10-2 (LP vs. SDP') 65
Table A.68.Regression Metrics for I-10-2' (LP vs. SDP') 65

LIST OF FIGURES

Figure 1.1. The plot of $X_{ii} = x_i^2$ along with the line passing through the points (l_i, l_i^2) and (u_i, u_i^2). 5

Figure 3.1. Performance Comparison with Derived Artificial Bounds 21
Figure 3.2. SDP Relaxation Performance in Instances with Hollow Matrices. 22
Figure 3.3. Diagram illustrating the calculation of statistic q for property p of matrices A_1 to A_m. 29

Figure 4.1. Confusion Matrices for Different Feature Designs. 38
Figure 4.2. Feature Importances for semi_dep ($n = 5$& $m = 1$). 39

1. INTRODUCTION

Quadratically constrained quadratic programs (QCQPs) are frequently encountered in optimization, appearing in fields such as operations research, engineering, and computer science. Comprehensive theoretical and computational research on QCQPs in the literature in recent years has greatly raised the efficiency and robustness of developed algorithms designed to address these problems. Also, there have been some recent advances in solvers, especially with Gurobi (Gurobi Optimization, 2021) and CPLEX (IBM Corp., 2022), involving substantial efficiencies in the solution of QCQP problems. Furthermore, Mosek (MOSEK ApS, 2020) has shown itself to be a dependable solver for general SDP problems crucial for effectively adressing QCQPs.

To formalize our approach, we consider the general formulation of a QCQP used throughout this study. Let A_i be real symmetric $n \times n$ matrices, b_i be real n-dimensional vectors, and c_i be real scalars for $i = 1, \ldots, m$. Additionally, let l and u be real n-dimensional vectors with $l \leq u$, where l and u can take values from the extended real numbers $(\mathbb{R} \cup \{\pm\infty\})$. Consider the following QCQP:

$$
\begin{aligned}
&(1.1a) && z = \inf_x \; x^T A_0 x + 2b_0^T x + c_0 && \\
&(1.1b) && \text{s.t. } x^T A_k x + 2b_k^T x + c_k \leq 0 && k = 1, \ldots, m \\
&(1.1c) && l \leq x \leq u. &&
\end{aligned}
$$

Despite their flexibility in modeling real-world problems, non-convex QCQPs are difficult to solve in practice due to their NP-hard nature (Anstreicher, 2009). Current literature has typically relied on either Semidefinite Programming (SDP) (Vandenberghe & Boyd, 1996) or Linear Programming (LP) relaxations (McCormick, 1976). Significant findings have shown that, under certain assumptions about the data or parameters in a random data model, the QCQP formulation of these problems has a tight SDP relaxation (Beck & Eldar, 2003; Wang & Kılınç-Karzan, 2022; Ye &

Zhang, 2003; Zhang, 2000), with further examples of exactness results available in Luo, Ma, So, Ye & Zhang (2010). As well as these SDP results, a number of recent papers also investigate conditions under which LP relaxations can be exact. More precisely, Qiu & Yıldırım (2023) recently established a necessary and sufficient condition on instances of QP that admit an exact Reformulation-Linearization Technique (RLT), showcasing the potential of LP relaxations in solving QCQPs under certain conditions.

For a user seeking the best relaxation technique for a QCQP instance that does not fit into a specific subclass, choosing between SDP and LP can be quite ambiguous. Because, each relaxation method works well for specific subproblems but often fails to address others effectively, without a complete understanding. For example, the SDP relaxation is particularly well-suited for problems such as the continuous relaxations of the Stable Set and MAXCUT problems in graph theory (Gaar & Rendl, 2020; Goemans & Williamson, 1995), Optimal Power Flow (Bai, Wei, Fujisawa & Wang, 2008; Kocuk, Dey & Sun, 2016), and Optimal Transmission Switching problems in electric power systems, as well as various problems in signal processing. On the other hand, problems that tend to favor LP relaxation include the Pooling Problem in chemical engineering (Marandi, Dahl & de Klerk, 2018), Circle Packing, and Layout problems (Khajavirad, 2024). This lack of understanding makes "structure-blind" general-purpose methods less effective than problem-specific solution approaches.

The goal of this research is to provide a "structure-aware" relaxation selection process for non-convex QCQPs that can determine whether an SDP- or LP-based approach is more advantageous based on the instance structure. Our approach is inspired by several studies that aim to predict the best optimization strategy for a given problem instance. For instance, Ghaddar, Gómez-Casares, González-Díaz, González-Rodríguez, Pateiro-López & Rodríguez-Ballesteros (2022) utilize learning techniques for spatial branching to increase the performance of a RLT for polynomial optimization problems (PO). Similarly, González-Rodríguez, Alvite-Pazó, Alvite-Pazó, Ghaddar & Díaz (2022) present a machine learning approach to predict the best performing conic constraints for strengthening RLT relaxations of a PO problem. The selection of input variables in these studies closely resembles our feature design approach, using number of variables, number of constraints, and features related to the graph representations of PO. Furthermore, Weiner, Ernst, Li & Sun (2023) explores the role of machine learning (ML) prediction quality of mixed integer programming (MIP) decompositions created via constraint relaxation. In line with these efforts, Kruber, Lübbecke & Parmentier (2017) propose a supervised learning approach to decide whether or not a reformulation should be applied to a

MIP, and which decomposition to choose when several are possible. All these efforts exemplify the promising synergy between optimization and machine learning, which has become a vibrant area of research. However, the studies most closely related to our work are the initial study by Bonami, Lodi & Zarpellon (2017) and its subsequent revision by Bonami, Lodi & Zarpellon (2022). These studies address the question of whether to linearize or not for convex mixed-integer quadratic programming problems. The machine learning models they employ, the exploitation of spectral properties of data matrices in the feature design phase, and the definition of custom performance metrics share similarities with our study and have been enlightening.

Before proceeding further, we establish some notations that will be used throughout this work.

Notation 1. *To denote a symmetric $n \times n$ matrix X, where $n \in \mathbb{Z}^+$, as positive semidefinite, we use $X \succeq 0$. Similarly, $X \succ 0$ indicates that the $n \times n$ matrix X is positive definite.*

Notation 2. *The Frobenius inner product of two $n \times n$ matrices X and Y is represented by $X \bullet Y$ and is calculated as $\sum_{i,j=1}^{n} X_{ij} Y_{ij}$ where $n \in \mathbb{Z}^+$.*

Notation 3. *The standard basis vector in $\mathbb{R}^n$, where $n \in \mathbb{Z}^+$, with 1 in the i-th position and 0 elsewhere is denoted by e_i.*

To illustrate the main relaxations, we consider the lifted reformulation of the QCQP introduced earlier in (1.1) which is a technique used to handle the non-convexity of the original problem (1.1). It introduces additional variables and constraints, resulting in a problem that is easier to relax.

In the lifted reformulation, we introduce a new matrix variable X which is intended to represent the outer product xx^T. This reformulation transforms the original non-convex quadratic constraints into linear constraints in terms of the new variables x and X. The resulting problem is as follows:

$$
\begin{aligned}
&(1.2a) \qquad && z = \inf_{x,X} \; A_0 \bullet X + 2b_0^T x + c_0 && \\
&(1.2b) \qquad && \text{s.t. } A_k \bullet X + 2b_k^T x + c_k \leq 0 && k = 1, \ldots, m \\
&(1.2c) \qquad && X = xx^T && \\
& && (1.1c). &&
\end{aligned}
$$

The constraint $X = xx^T$ ensures that X is a rank-one matrix, preserving the non-convex nature of the original problem, and we can derive various convex relaxations of the QCQP (1.2) by relaxing this constraint.

The LP relaxation of of the lifted QCQP (1.2), replaces the non-convex constraint $X = xx^T$ with a set of linear inequalities known as McCormick envelopes, which are used to approximate the bilinear terms $x_i x_j$:

$$
\begin{aligned}
&(1.3a) && z_{\text{LP}} = \inf_{x,X} \; A_0 \bullet X + 2b_0^T x + c_0 && \\
& && \text{s.t. } (1.2b) && \\
&(1.3b) && X_{ij} - l_j x_i - l_i x_j + l_i l_j \geq 0 && 1 \leq i \leq j \leq n \\
&(1.3c) && X_{ij} - u_j x_i - l_i x_j + l_i u_j \leq 0 && 1 \leq i \leq j \leq n \\
&(1.3d) && X_{ij} - l_j x_i - u_i x_j + u_i l_j \leq 0 && 1 \leq i \leq j \leq n \\
&(1.3e) && X_{ij} - u_j x_i - u_i x_j + u_i u_j \geq 0 && 1 \leq i \leq j \leq n.
\end{aligned}
$$

Similarly, the SDP relaxation of the lifted reformulation (1.2) replaces the non-convex constraint $X = xx^T$ with a positive semidefinite constraint on an augmented matrix Y:

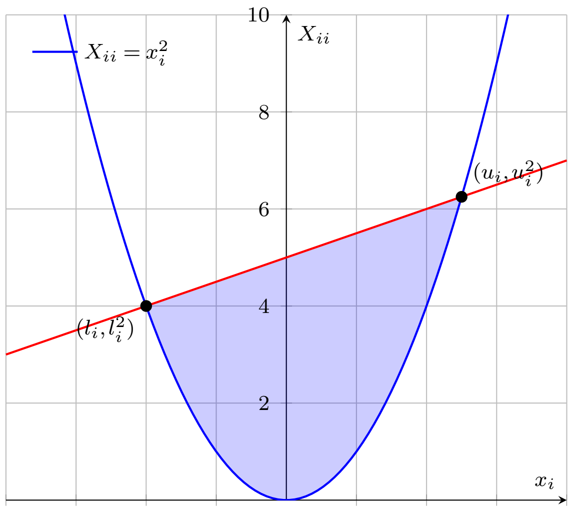
$$
\begin{aligned}
&(1.4a) && z_{\text{SDP}} = \inf_{x,X,Y} \; A_0 \bullet X + 2b_0^T x + c_0 \\
& && \text{s.t. } (1.1c), (1.2b) \\
&(1.4b) && Y := \begin{bmatrix} X & x \\ x^T & 1 \end{bmatrix} \succeq 0
\end{aligned}
$$

If $A_0, \ldots, A_m$ are all positive semidefinite, then the problem is convex because both the objective function and the constraints are convex functions. Nevertheless, when these matrices are neither positive nor negative semidefinite, the problem remains non-convex, leading to considerable difficulties in finding global optima.

These relaxations transform the original non-convex QCQP (1.1) into convex optimization problems that can be solved very effectively and reliably using standard optimization techniques.

The SDP relaxation provided in (1.4) does not utilize the variable bounds effectively. We propose adding additional constraints to create an enhanced SDP relaxation, denoted as *SDP'*. Specifically, we introduce constraints that relate to the graph of $X_{ii} = x_i^2$ (see Figure 1.1).

Figure 1.1 The plot of $X_{ii} = x_i^2$ along with the line passing through the points (l_i, l_i^2) and (u_i, u_i^2).



The area under the line passing through the bounds l_i and u_i intersects the points l_i^2 and u_i^2. Given these two points, the equation of the line can be determined and used to define the additional constraints. These constraints can be expressed as:

$$X_{ii} \leq (u_i + l_i)x_i - u_i l_i \quad \text{for } i = 1, \ldots, n.$$

The general formulation of SDP', the tighter SDP relaxation created by incorporating these additional constraints, is as follows:

$$\begin{aligned} &(1.5\text{a}) && z_{\text{SDP}} = \inf_{x,X,Y} \; A_0 \bullet X + 2b_0^T x + c_0 \\ & && \text{s.t. } (1.1\text{c}), (1.2\text{b}), (1.4\text{b}) \\ &(1.5\text{b}) && X_{ii} \leq (u_i + l_i)x_i - u_i l_i \qquad i = 1, \ldots, n. \end{aligned}$$

This research aims to explore the theoretical underpinnings of these relaxations by examining the spectral properties and sparsity patterns of the data matrices. The ultimate goal is to develop a classification model that can accurately predict whether a given QCQP instance is SDP-favoring or LP-favoring.

We state the research question as follows: How can one unravel the structure of an unknown QCQP instance to be either SDP-favoring or LP-favoring, i.e., whether the

SDP or the LP relaxation would produce the stronger bound without solving them? Our approach relies on theoretical exploration of the instances by analyzing the effect of the spectral properties and sparsity pattern of the data matrices to the feasible regions of the aforementioned relaxations. This approach also utilizes a classification model that correctly predicts the category of a new instance (either SDP-favoring or LP-favoring). We handle all aspects of the learning process, including dataset generation, feature design, and labeling procedures.

Our findings indicate significant differences in the performance of SDP and LP relaxations based on the structural properties of the QCQP instances. Notably, SDP relaxations tend to perform better in instances with convex constraints, while LP relaxations are more effective in reverse convex scenarios. We also observe that in instances with hollow matrices, SDP relaxations often become unbounded, highlighting their inadequacy in handling certain structures.

The remainder of this thesis is structured as follows. Chapter 2 details the process of generating the QCQP instances analyzed in this study. In Chapter 3, we describe our methodology, beginning with an exploratory data analysis and progressing to three distinct feature engineering approaches for the machine learning models. Subsequently, we introduce classification and regression models to categorize QCQP instances according to their relations to LP and SDP relaxations. Chapter 4 contains the empirical results of the learning experiments conducted in this research. Finally, in Chapter 5 we present the conclusions of the study.

2. INSTANCE GENERATION

In this chapter, we describe the process of generating QCQP instances, which is pivotal for our subsequent analyses. Our primary objective is to generate diverse dataset to study how different properties of the instance structure affect the success of relaxations. We hypothesize that specific instance structures, such as the number of variables (n), the number of constraints (m) (Ghaddar et al., 2022; González-Rodríguez et al., 2022), matrix rank, and the existence of finite variable bounds, are important factors to consider. Additionally, the effect of the convex-concave nature of the problem has been discussed in previous works by Fu, Luo & Ye (1998), Nemirovskii, Roos & Terlaky (1999), and Nesterov (2000), and the presence of a single reverse convex constraint (Hillestad & Jacobsen, 1980).

Furthermore, the success of relaxations may be influenced by various factors, including the spectral properties, as discussed in Bonami, Lodi & Zarpellon (2022). Other factors include the presence of bilinear constraints (McCormick, 1976; Torres, 1990) and sparsity patterns (such as complete, bipartite, tree, forest, cycle, chordal, and planar graphs), as well as the diagonal nature of matrices (Burer & Ye, 2018; Kim & Kojima, 2003). We aim to create a sufficiently diverse dataset so that we can systematically observe the effects of these instance structures. Section 2.1 focuses on matrices that exhibit various sparsity patterns. In this section, we use the `Python NetworkX` package (Hagberg, Schult & Swart, 2008) to create and manipulate graphs, including operations such as adding or removing nodes and edges. In Section 2.2, we discuss the generation of $n \times n$ random symmetric matrices with controlled properties. Additionally, Section 2.3 covers QCQPs that do not have finite variable bounds. Throughout our matrix generation process, we often use random numbers chosen from specific distributions.

Notation 4. *The symbol $\mathcal{U}(a,b)$ denotes a uniform distribution with bounds a and b. For instance, a random variable X that is uniformly distributed between -2 and 2 is written as:*

$$X \sim \mathcal{U}(-2,2)$$

2.1 Matrices with Different Sparsity Patterns

In this section, we describe the generation of various types of $n \times n$ matrices used in our study. These matrices include symmetric hollow, bipartite, tree, planar, chordal, and diagonal forms.

2.1.1 Hollow Matrices

To generate $n \times n$ symmetric hollow matrices, we set the diagonal elements to zero and fill the off-diagonal elements with random values from a uniform distribution. The resulting matrix M is produced using Algorithm 1.

Algorithm 1 Hollow Matrix Generation.

```
Input: $n$
Output: $M$: Symmetric $n \times n$ hollow matrix
1: Generate a symmetric $n \times n$ matrix $M$ with elements sampled from $\mathcal{U}(-2,2)$
2: Set all diagonal elements of $M$ to 0
3: return $M$
```

2.1.2 Bipartite Graph Matrices

We use the `bipartite` module of the `networkx` package to generate random bipartite graphs. First, we construct a bipartite graph G with k nodes in one partition and $n-k$ nodes in the other partition. The adjacency matrix A of graph G is computed. Afterwards, A is converted into a distance matrix D, in which each edge (with a value of 1) is substituted with a random integer, indicating the distance between nodes. The latter two procedures are applicable to all subsequent graph types.

Algorithm 2 Bipartite Matrix Generation.

```
Input: $n$
Output: $D$: An $n \times n$ bipartite distance matrix
1: Randomly choose $k$ such that $2 \leq k \leq n-2$
2: Randomly choose $e$ such that $2 \leq e \leq k \times (n-k)$
3: Generate a bipartite graph $G$ with $k$ and $n-k$ nodes and $e$ edges
4: Compute the adjacency matrix $A$ of graph $G$                ▷ Common Step 1
5: Convert adjacency matrix $A$ into distance matrix $D$ with random weights    ▷
   Common Step 2
6: return $D$
```

2.1.3 Random Spanning Tree Matrices

Tree matrices are generated from random spanning trees. Similar to the bipartite matrix, the adjacency matrix is converted into a distance matrix with randomly assigned weights. The steps are as follows:

Algorithm 3 Random Tree Matrix Generation.

```
Input: $n$
Output: $D$: An $n \times n$ random tree distance matrix
1: Generate a random tree $T$ with $n$ nodes
2: Call Algorithm 2, Steps 4 and 5 respectively with input $T$
3: return $D$
```

2.1.4 Planar Graph Matrices

To generate a planar matrix, we first create a complete graph G with n nodes. Then, with a brute force technique, we remove random edges until we obtain a planar structure. We check whether the graph we produce is connected, if not, we add a random edge. In the last stage, we perform the common steps, which are producing the adjacency matrix and distance matrix respectively from G.

Algorithm 4 Generate Planar Matrix.

```
Input: $n$
Output: $D$: An $n \times n$ planar distance matrix
 1: Generate a complete graph $G$ with $n$ nodes
 2: Shuffle the edges of $G$
 3: Set $is\_planar \leftarrow$ False
 4: while not $is\_planar$ do
 5:     Remove an edge from $G$
 6:     Check if $G$ is planar
 7: end while
 8: if $G$ is not connected then
 9:     Add an edge between two random nodes in $G$
10: end if
11: Call Algorithm 2, Steps 4 and 5 respectively with input $G$
12: return $D$
```

2.1.5 Chordal Graph Matrices

First, we generate a random tree T consisting of n nodes. In order to guarantee that the graph is chordal, we iterate over each node in T. For any node that has at least two neighboring nodes, we randomly choose two neighbors, denoted as v and w, and add an edge between v and w to form a chord. These methods guaranteed that the graph retained its chordal property. The latter steps consist of transforming the adjacency matrix into a distance matrix, a process that applies to all types of matrices previously mentioned.

Algorithm 5 Generate Chordal Matrix.

```
Input: $n$
Output: $D$: An $n \times n$ chordal distance matrix
 1: Generate a random tree $T$
 2: for each node in $T$ do
 3:     Get the list of neighbors of the current node
 4:     if the number of neighbors is at least 2 then
 5:         Randomly select two neighbors $v$ and $w$ from *neighbors*
 6:         Add an edge between $v$ and $w$ to create a chord
 7:     end if
 8: end for
 9: Call Algorithm 2, Steps 4 and 5 respectively with input $T$
10: return $D$
```

2.1.6 Conversion of Adjacency to Distance Matrices

For bipartite, tree, planar, and chordal matrices, the adjacency matrices are converted to distance matrices. Each edge in the adjacency matrix is replaced with a random integer, ensuring symmetry and adding variability to the weights. The method is as follows:

$$D[i,j] = \begin{cases} R_{ij}, & \text{if } A[i,j] = 1 \\ 0, & \text{otherwise,} \end{cases}$$

where R_{ij} is a randomly selected integer $\sim \mathcal{U}(-10,10)$. To illustrate matrices with different sparsity patterns, we present examples below.

$$\begin{bmatrix} 4 & 0 & 0 & -10 \\ 0 & 7 & 4 & -9 \\ 0 & 4 & 6 & 0 \\ -10 & -9 & 0 & 8 \end{bmatrix}$$

(a) A bipartite matrix.

$$\begin{bmatrix} 0 & 4.94 & -0.32 & 2.31 \\ 4.94 & 0 & 1.04 & -4.16 \\ -0.32 & 1.04 & 0 & 0.47 \\ 2.31 & -4.16 & 0.47 & 0 \end{bmatrix}$$

(b) A hollow matrix.

Next, we describe the generation of various types of $n \times n$ diagonal matrices used in our study. These matrices include those with ordered and randomly placed 1s and

-1s, as well as diagonal matrices with random values. In all cases, the resulting matrix D is a diagonal matrix formed as $D = \texttt{diag}(v)$, where v is a vector defined according to the specific method.

2.1.7 Diagonal Matrices

In this section, we produce diagonal matrices that vary in order and magnitude of their eigenvalues. The number of negative eigenvalues for these matrices, n', can be specified or randomly determined. First, we create ordered diagonal matrices with eigenvalues -1 and 1, , where n' values are -1 followed by $n - n'$ values of 1. In the second method, we give a random order to -1s and 1s. And finally, we produce diagonal matrices with eigenvalues containing random numbers in random positions, provided that (n') of them are negative. The aim here is to enhance the diversity of the matrices used in our tests.

Algorithm 6 Diagonal Matrix Generation.

```
Input: $n$, $type$ (one of {'ordered_ones', 'random_ones', 'random_randnums'}), $n'$ (number of negative eigenvalues, optional)
Output: $D$: An $n \times n$ diagonal matrix based on the specified type
1: if $n'$ is not specified then
2:     Set $n'$ to a random integer between 0 and $n$ (inclusive)
3: end if
4: Initialize vector $v$ of length $n$
5: if $type$ is 'ordered_ones' then
6:     Set the first $n'$ elements of $v$ to -1, and the remaining $(n - n')$ elements to 1
7: else if $type$ is 'random_ones' then
8:     Set all elements of $v$ to 1
9:     Select $n'$ random positions from the range 0 to $n - 1$ and store them in $neg\_list$
10:    for each index $idx$ in $neg\_list$ do
11:        Set the element of the vector $v$ at position $idx$ to -1
12:    end for
13: else if $type$ is 'random_randnums' then
14:    Generate $n$ random values from the range 1 to 10 and store them in $v$
15:    Select $n'$ random positions from the range 0 to $n - 1$ and store them in $neg\_list$
16:    for each index $idx$ in $neg\_list$ do
17:        Set the element of the vector $v$ at position $idx$ to the negative of its current value
18:    end for
19: end if
20: Form a diagonal matrix $D$ from vector $v$
21: return $D$
```

These three methods allowed us to create a diverse set of diagonal matrices, each with unique structural properties. Below, we provide two examples:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

(c) A diagonal matrix with ±1 eigenvalues.

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & -6 & 0 \\ 0 & 0 & -7 \end{bmatrix}$$

(d) A diagonal matrix with random eigenvalues.

2.2 Random Matrices

This part of the study describes the generation of various types of $n \times n$ random matrices used in our study. These include random symmetric matrices, matrices with specific eigenvalue properties, and random symmetric positive definite matrices.

2.2.1 Random Symmetric Matrix

To generate random symmetric matrices, we began by creating a matrix A with elements drawn randomly $\sim \mathcal{U}(-5,5)$.

Algorithm 7 Random Symmetric Matrix Generation.

```
Input: $n$
Output: $M$: An $n \times n$ random symmetric matrix
1: Generate a random symmetric $n \times n$ matrix $M$ with elements sampled from
   $\mathcal{U}(-5,5)$
2: return $M$
```

2.2.2 Random Matrices with Specified Eigenvalues

Random matrices with specified eigenvalues are generated using a procedure involving QR decomposition. For matrices with eigenvalues of ± 1, first we generate random matrix, and perform its QR decomposition to obtain an orthogonal matrix Q. Then, we form a diagonal matrix Λ with entries of 1 and -1, and the final matrix M is constructed as $M = Q\Lambda Q^T$, representing its eigendecomposition. In the case of matrices with random eigenvalues, the process follows the same initial steps, but the diagonal matrix Λ is composed of random numbers selected from a specified range rather than fixed entries of 1 and -1.

Algorithm 8 Random Matrix with Specified Eigenvalues.

Input: n, *type* (either 'ones' or 'random'), n' (optional)

Output: M: Random $n \times n$ matrix with specified eigenvalues

```
1: if $n'$ is not specified then
2:     Randomly select $n'$ from 0 to $n$
3: end if
4: Generate a random $n \times n$ matrix $rand\_mtrx$ with entries as continuous random
   numbers $\sim \mathcal{U}(0, 9)$
5: Generate an orthogonal matrix $Q$
6: Initialize vector $v$ of length $n$
7: if $type$ is 'ones' then
8:     Fill $v$ with ones
9:     Randomly select $n'$ positions in $v$ and set them to -1
10: else if $type$ is 'random' then
11:     Generate $v$ with random integers between 1 and 10
12:     Randomly select $n'$ positions in $v$ and set them to -1
13: end if
14: Form a diagonal matrix $\Lambda$ using the vector $v$
15: Construct the matrix $M$ from $\Lambda$ and $Q$
16: return $M$
```

These methods ensure the desired eigenvalue constraints or diversity.

2.2.3 Random Symmetric Positive Definite Matrix

Random symmetric $n \times n$ positive definite matrices are generated using a standard procedure that ensures positive definiteness. The `scikit-learn` library (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, Vanderplas, Passos, Cournapeau, Brucher, Perrot & Duchesnay, 2011), a powerful tool for machine learning in Python, provides the function `make_spd_matrix` to generate symmetric positive definite (SPD) matrices.

These four methods allowed us to generate diverse sets of data, some reflecting unique structural properties. Below are some illustrative examples for $n = 3$ to demonstrate the random data matrix generation process:

$$\begin{bmatrix} 0.526 & -0.145 & 0.837 \\ -0.145 & 0.955 & 0.257 \\ 0.837 & 0.257 & -0.481 \end{bmatrix}$$

(e) A random matrix with eigenvalues of ± 1.

$$\begin{bmatrix} 0.722 & 0.506 & -0.498 \\ 0.506 & 1.856 & -1.496 \\ -0.498 & -1.496 & 1.554 \end{bmatrix}$$

(f) A random symmetric, positive-definite matrix.

2.3 QCQPs without Finite Variable Bounds

In addition to the examples presented above, we seek to examine problem types where no finite variable bounds are imposed in the original problem formulation. Nevertheless, for these examples, variable limits are still necessary to construct the LP Relaxation. To address this, we generate examples featuring at least one strictly convex constraint. We derive artificial boundaries, by determining the smallest axis-parallel bounding box that encloses the ellipsoid defined by this constraint. We employ Lagrangian multipliers to derive the closed-form solution for the Quadratically Constrained Program (QCP) described in problem (2.1).

$$z_{\text{QCP}} = \min \backslash \max_{x} \; e_i^T x \tag{2.1a}$$

$$\text{s.t. } \frac{1}{2} x^T A x + b^T x + c \leq 0 \quad (\text{where } A \succ 0). \tag{2.1b}$$

The optimal solution x^* is given by:

$$x^* = -A^{-1}\left(\frac{\mathbf{e}_i}{\lambda} + \mathbf{b}\right) \tag{2.2}$$

where λ is the Lagrange multiplier. By substituting x^* into the objective function, we obtain the optimal value:

$$z_{\text{QCLP}} = \mathbf{e}_i^\top x^* \tag{2.3}$$

$$= \mathbf{e}_i^\top \left(-A^{-1}\left(\frac{\mathbf{e}_i}{\lambda} + \mathbf{b}\right)\right) \tag{2.4}$$

$$= -\mathbf{e}_i^\top A^{-1}\left(\frac{\mathbf{e}_i}{\lambda} + \mathbf{b}\right). \tag{2.5}$$

To calculate the values of λ, we define:

$$p = -\frac{1}{2}\mathbf{b}^\top A^{-1}\mathbf{b} + c, \tag{2.6}$$

$$r = \frac{1}{2}(\mathbf{e}_i^\top A^{-1}\mathbf{b} - \mathbf{b}^\top A^{-1}\mathbf{e}_i), \tag{2.7}$$

$$s = \frac{1}{2}\mathbf{e}_i^\top A^{-1}\mathbf{e}_i. \tag{2.8}$$

The discriminant Δ is given by:

$$\Delta = r^2 - 4ps. \tag{2.9}$$

The roots λ_{min} and λ_{max} are:

$$\lambda_{\text{min}} = \frac{-r - \sqrt{\Delta}}{2p}, \tag{2.10}$$

$$\lambda_{\text{max}} = \frac{-r + \sqrt{\Delta}}{2p}. \tag{2.11}$$

The bounds for x_i are obtained by evaluating the objective function at λ_{min} and λ_{max}. Specifically, substituting λ_{min} yields the lower bound, $\min(z_{\text{QCLP}})$, and substituting λ_{max} yields the upper bound, $\max(z_{\text{QCLP}})$.

3. RESEARCH METHODOLOGY

This chapter summarizes the systematic approaches and techniques used for comparative analysis of the two main relaxations of QCQP. Initially, in Section 3.1, we use various optimization modeling languages to solve QCQP problems and related relaxations, which lays a foundation for our subsequent analyses. Next, we examine QCQP instances to derive conjectures and insights from the exploratory analysis in Section 3.2. We conduct an empirical analysis by applying machine learning techniques with various feature generation methods in Section 3.3. We generate features through three distinct methods, each demonstrating a unique relationship between the number of features and the dimensionality of the problem.

3.1 Optimization Modeling Languages and Related Solvers

Within the scope of this project, the first step toward evaluating the relative efficacy of QCQP relaxations is to attain the lower bounds provided by these relaxations. As depicted in the Table 3.1, it is noteworthy that not every type of problem, such as Linear Programming (LP), Quadratic Programming (QP), Second-Order Cone Programming (SOCP), Exponential Programming (EXP), Semidefinite Programming (SDP), and Mixed-Integer Programming (MIP), may be supported by every modeling language and associated solver. To address this, we obtain exact solutions of QCQP problems using Gurobi Optimizer (Gurobi Optimization, 2021), while employing Python software packages such as CVXPY (Diamond & Boyd, 2016) and Mosek (MOSEK ApS, 2020) for SDP and LP relaxations.

Table 3.1 Supported Problem Types by Various Optimization Software Packages.

	LP	**QP**	**SOCP**	**EXP**	**SDP**	**POW**	**MIP**
CPLEX	✓	✓	✓				✓
CVXPY	✓	✓	✓	✓	✓	✓	
Gurobi	✓	✓	✓				✓
MOSEK	✓	✓	✓	✓	✓	✓	✓*

(*) Except mixed-integer SDP.

3.2 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is the cornerstone of the learning-based approach in Section 3.3. Herein, we seek to understand the key features underlying the data that may affect the predictive outcome. In this section, we use EDA and data-driven methods to examine relationships between the structural properties of the QCQP instances and related relaxations.

Table 3.2 Distribution of Data: LP-favoring vs. SDP-favoring for Different Number of Decision Variables and Constraints.

n	m	**LP-favoring**	**SDP-favoring**	**Total**
5	1	32454	17546	50000
5	2	31657	18343	50000
10	1	33151	16849	50000
10	2	32846	17154	50000

Table 3.2 presents the distribution of LP-favoring and SDP-favoring instances for different number of decision variables ($n = 5$ and $n = 10$) and numbers of constraints ($m = 1$ and $m = 2$). The data indicates that for every type of data configuration, whether $n = 5$ or $n = 10$, and whether $m = 1$ or $m = 2$, the distributions of LP-favoring and SDP-favoring instances are reasonably balanced.

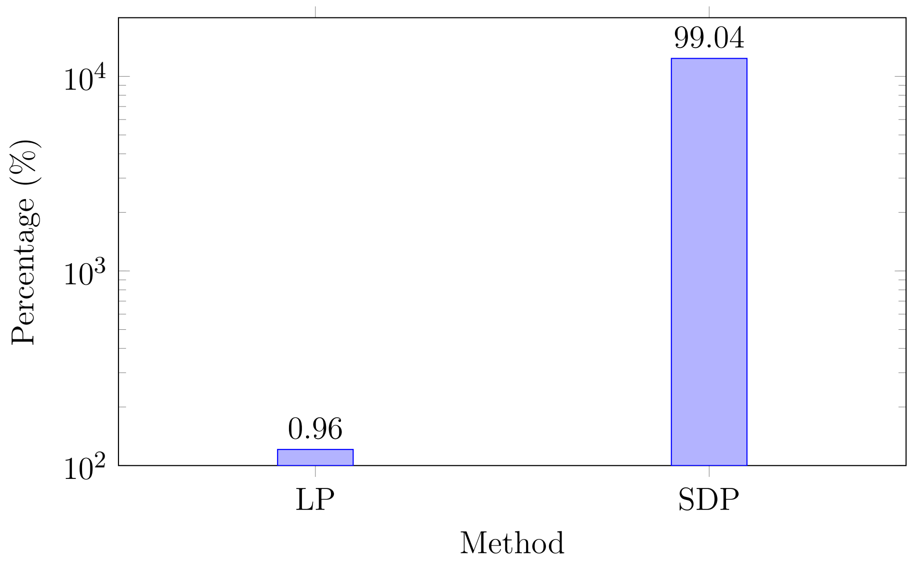
Table 3.3 Percentages of Different Metrics for A_1 and A_2.

Constraint Type	A_1	A_2
% of Convex Constraints	49.80%	9.56%
% of Reverse Convex Constraints	0.56%	10.58%
% of Diagonal Constraints	14.91%	24.44%
% of Hollow Constraints	4.10%	9.13%
% of Bipartite Constraints	6.70%	6.52%
% of Tree Constraints	6.83%	6.47%
% of Chordal Constraints	6.39%	7.01%
% of Planar Constraints	6.05%	7.48%

Table 3.3 provides a summary of the distribution of different instance types for the initial constraint (A_1) and the subsequent constraint (A_2). In this study, we shall refer to a problem as "reverse convex" if all data matrices are negative-definite. The primary constraint is predominantly based on convex constraints, as the data is produced in a way that the more convex constraints come before the less convex constraints. The second constraint has a substantially greater percentage of diagonal constraints (24.44%) compared to the first constraint (14.91%). The sparsity patterns, such as bipartite, planar, chordal, and tree, have nearly identical distributions in both A_1 and A_2, with minor discrepancies that have negligible effects on the overall structural comparison. The contrasts between the first and second constraints are highlighted by their differing structural properties.

Figure 3.1 displays an evaluation of the effectiveness of LP and SDP relaxations when the artificial bounds are introduced. As described in Section 2.3, this approach includes incorporating bounds generated from convex constraints into the original QCQP problem, which does not have intrinsic finite variable bounds. The empirical data shows that the LP relaxation is superior to the SDP relaxation in just 0.96% of cases, whereas the SDP relaxation is considerably superior in 99.04% of cases. The significant disparity indicates that the LP relaxation is significantly insufficient in this context, yielding a considerably lower optimal value and therefore, the LP relaxation fails to offer a meaningful bound.

Figure 3.1 Performance Comparison with Derived Artificial Bounds



As a result of these observations, we propose the following conjecture:

Conjecture 1. *If the original problem does not have any finite variable bounds, artificial bounds are introduced, then with high probability $z_{LP} \leq z_{SDP}$.*

Figure 3.2 illustrates the performance of SDP relaxation in a scenario where the data matrices in the QCQP are hollow matrices (zero diagonal). The analysis reveals that in all observed instances, the SDP relaxation becomes unbounded. This finding indicates that the SDP relaxation fails completely under these conditions, highlighting its inadequacy in handling instances with zero diagonal data matrices.

Based on these observations, we propose another conjecture:

Conjecture 2. *If A_i matrices are hollow, then $z_{SDP} \leq z_{LP}$.*

Figure 3.2 SDP Relaxation Performance in Instances with Hollow Matrices.

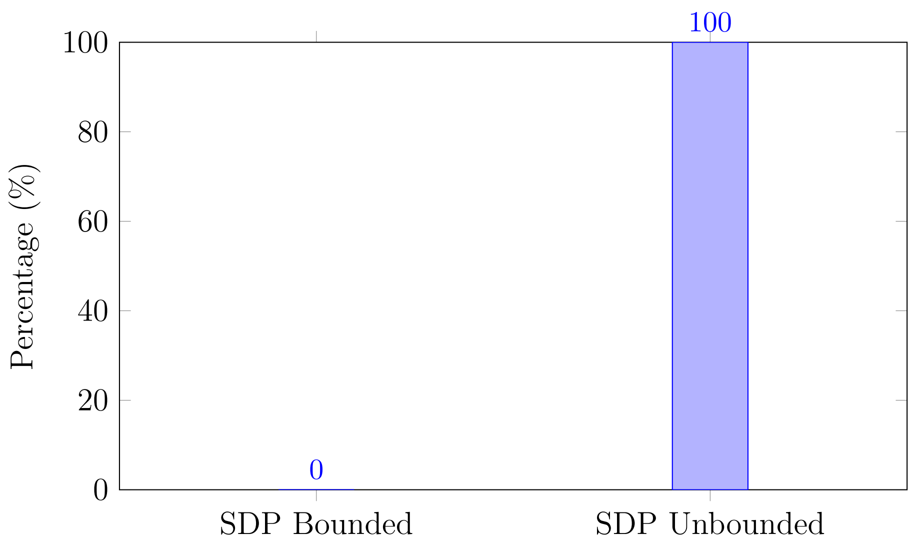


Table 3.4 offers an overview of the LP and SDP relaxations' performance with respect to various problem types and objectives. The data shows the superiority of SDP relaxation by a wide margin for convex problems and objectives compared to LP relaxation. Conversely, LP relaxation is shown to be more effective for reverse convex problems. This general trend underscores the importance of choosing the suitable relaxation method based on the specific characteristics of the problem at hand to achieve optimal performance.

Table 3.4 Comparison of LP and SDP Relaxations for Convex and Reverse Convex Problems.

Problem Type	Outperforming Relaxation	
	LP	SDP
Convex Problem	0.19%	99.80%
Reverse Convex Problem	100%	0%
Convex Objective	12.55%	87.43%
Concave Objective	93.03%	6.96%

From the results shown in Table 3.4, we may formulate two conjectures about the performance of LP and SDP relaxations in relation to the spectral properties of A_0, which impact the convexity or concavity of the objective function. The table demonstrates a distinct trend in which SDP relaxations exhibit outstanding performance in scenarios involving convex problems, but LP relaxations are more effective for reverse convex problems. Based on these observations, the following conjectures can be formulated:

Conjecture 3. *If A_0 is positive semi-definite, i.e. the objective function is convex, then with high probability $z_{LP} \leq z_{SDP}$.*

Conjecture 4. *If A_0 is negative semi-definite, i.e. the objective function is concave, then with high probability $z_{SDP} \leq z_{LP}$.*

Although we have not yet been able to prove our conjectures, we are actively working on creating a family of counterexamples and our efforts to develop a suitable probabilistic model for our conjectures continue.

3.3 Learning Experiments

This section outlines the supervised learning techniques utilized in this study. We develop classification and regression models to analyze how QCQP instances relate to LP and SDP relaxations, but a QCQP sample cannot be given as input to machine learning as in the form 1.1. Therefore, we turn QCQP instances into a meaningful set of input in Section 3.3.1, and we tag the data so that the machine can learn in Section 3.3.2. Finally, Section 3.3.3 gives provides information about the training and testing process.

3.3.1 Feature Design

After generating QCQP instances and conducting exploratory analysis, the last step of our data-driven technique is feature extraction for our machine learning models. These features cover a range of properties of the data matrices that are expected to have a substantial impact on the performance of the studied relaxations, including the negative eigenvalues and the sparsity patterns in the data matrices. At first, our method of feature engineering in Section 3.3.1.1 largely depended on features that were highly dependent on m and n. Nevertheless, we then identified the potential limitations of our method, particularly its lack of generalizability and its tendency to become explode when dealing with larger matrix sizes and instances that involve a significant number of constraints. In order to tackle these difficulties, we modified our approach to ensure that the number of features remained unaffected, first by the decision variables (as explained in Section 3.3.1.2), and then by the total instance size (as discussed in Section 3.3.1.3).

3.3.1.1 Fully Dimension-Dependent Setup

At this stage of the study, the feature set is directly linked to the dimensionality of the decision variables and the number of constraints. For instance, each eigenvalue is represented as a distinct feature, and the number of negative eigenvalues or rank can be quantified as an integer ranging from 0 to n.

Notation 5. *Let A be a $n \times n$ square matrix, where $n \in \mathbb{Z}^+$. The eigenvalues of A are denoted by $\lambda(A)$, which is a vector of eigenvalues sorted from the largest to the smallest.*

Definition 1. *Let $x \in \mathbb{R}^n$ be a vector, where $n \in \mathbb{Z}^+$. We define $\eta(x)$ as the number of negative values in x:*

$$\eta(x) = \sum_{i=1}^{n} \mathbf{1}_{\{x_i<0\}},$$

where $\mathbf{1}_{\{x_i<0\}}$ is an indicator function that equals 1 if $x_i < 0$ and 0 otherwise.

By applying this function, $\eta(\lambda(A_i))$ specifically yields the number of negative eigenvalues for the corresponding matrix A_i.

Definition 2. *Let A be a $n \times n$ square matrix, where $n \in \mathbb{Z}^+$ and $\mathcal{P}_s(A) : \mathbb{R}^{n\times n} \to \{0,1\}$ be a function defined as follows:*

$$\mathcal{P}_s(A) = \begin{cases} 1 & \textit{if the sparsity pattern of } A \textit{ is } s \\ 0 & \textit{otherwise,} \end{cases}$$

where $s \in \mathcal{S} = \{$Diagonal, Hollow, Bipartite, Tree, Chordal, Planar$\}$.

As an example, defining this function allows $\mathcal{P}_D(A_i)$ to return 1 if A_i is a diagonal matrix and 0 otherwise.

Table 3.5 provides a detailed description of the features generated for the fully dimension-dependent setup. This approach generates features that are highly dependent on both the dimensionality of the decision variables (n) and the number of constraints (m). The spectral properties section includes features such as the number of negative eigenvalues ($\eta(\lambda(A_i))$), the jth largest eigenvalue ($\lambda_j(A_i)$), and the rank ($\rho(A_i)$) of each constraint matrix A_i. Additionally, the sparsity pattern section includes the feature $\mathcal{P}_s(A_i)$, which indicates whether a constraint matrix A_i has a specific sparsity pattern $s \in \mathcal{S}$. The *BoundsExist?* feature indicates whether the original problem has finite variable bounds or if the artificial bounds method are derived as described in Section 2.3. The total number of features is a function of both n and m, calculated as below.

Table 3.5 Description of Fully Dimension-Dependent Setup.

Notation	Description
Spectral properties	
$\eta(\lambda(A_i))$	Number of negative eigenvalues of A_i, $\quad i = 1, \ldots, m$.
$\lambda_j(A_i)$	jth largest eigenvalue of A_i, $\quad j = 1, \ldots, n$ and $i = 1, \ldots, m$.
$\rho(A_i)$	Rank of A_i, $\quad i = 1, \ldots, m$
Sparsity pattern	
$\mathcal{P}_s(A_i)$	1 if A_i has a sparsity pattern of $s \in \mathcal{S}$, 0 otherwise, $i = 1, \ldots, m$
BoundsExist?	1 if the original problem has finite variable bounds, 0 if artificial bounds method is used
Total Number of Features	$(n + \lvert\mathcal{S}\rvert + 2)(m + 1) + 1$

3.3.1.2 Semi Dimension-Dependent Setup

In this approach, the feature set is designed to be independent of n. Nevertheless, the number of features increases with the number of constraints m. This design strategy allows the features to encapsulate the essential characteristics of the constraints without being directly influenced by the size of the decision variable vector. Instead of providing all the eigenvalues, we define a ratio to represent the essential spectral properties of the constraints.

Definition 3. *The ratio of the sum of the absolute values of negative elements to* *the* 1*-norm of x in a nonzero vector $\mathbf{x} \in \mathbb{R}^n$ is denoted by $\theta(x)$. Mathematically, it is defined as*

$$\theta(x) = \frac{\sum_{i=1}^{n} |x_i| \mathbf{1}_{\{x_i<0\}}}{\|x\|_1},$$

where x_i represents the elements of the vector x, $\mathbf{1}_{\{x_i<0\}}$ is the indicator function that equals 1 if $x_i < 0$ and 0 otherwise, and $\|x\|_1$ is the 1*-norm of x, given by* $\|x\|_1 = \sum_{i=1}^{n} |x_i|$.

Table 3.6 Description of Semi Dimension-Dependent Setup.

Notation	Description
Spectral properties	
$\eta(\lambda(A_i))/n$	Number of negative eigenvalues of A_i over n, $i = 1,\ldots,m$.
$\theta(\lambda(A_i))$	Negative eigenvalue ratio of A_i, $i = 1,\ldots,m$
$\lambda_{\min}(A_i)$	Smallest eigenvalue of A_i, $i = 1,\ldots,m$.
$\lambda_{\max}(A_i)$	Largest eigenvalue of A_i, $i = 1,\ldots,m$.
$\rho(A_i)/n$	Rank of A_i over n, $i = 1,\ldots,m$.
Sparsity pattern	
$\mathcal{P}_s(A_i)$	1 if A_i has a sparsity pattern of $s \in \mathcal{S}$, 0 otherwise, $i = 1,\ldots,m$
BoundsExist?	1 if the original problem has finite variable bounds, 0 if artificial bounds are derived
Total Number of Features	$(\lvert\mathcal{S}\rvert + 5)(m+1) + 1$

3.3.1.3 Dimension-Independent Setup

Our final approach to feature engineering involves generating features whose number is constant, irrespective of the problem size. This method is particularly advantageous for large-scale problems with a substantial number of variables and constraints. The main idea of this approach is that rather than providing individual information for each constraint, we introduce aggregate statistics of all constraints.

Notation 6. *Let $\Xi, \Theta, \Lambda_{\min}, \Lambda_{\max}, \varrho, \Phi_s$ be vectors of size m, where $m \in \mathbb{Z}^+$, defined as follows:*

$$\Xi = \left[\eta(\lambda(A_1))/n \quad \eta(\lambda(A_2))/n \quad \dots \quad \eta(\lambda(A_m))/n\right]^T.$$

Here, $\eta(\lambda(A_i))$ denotes the number of negative eigenvalues of the matrix A_i, and n represents normalization by n, the size of the decision variable. This vector thus captures the proportion of negative eigenvalues for each constraint matrix A_i.

$$\Theta = \left[\theta\left(\lambda(A_1)\right) \quad \theta\left(\lambda(A_2)\right) \quad \dots \quad \theta\left(\lambda(A_m)\right)\right]^T.$$

Here, $\theta\left(\lambda(A_i)\right)$ denotes the ratio of the sum of the absolute values of negative eigenvalues to the sum of the absolute values of all eigenvalues for the matrix A_i. This vector captures the negative eigenvalue ratios for each constraint matrix A_i.

$$\Lambda_{\min} = \left[\lambda_{\min}(A_1) \quad \lambda_{\min}(A_2) \quad \dots \quad \lambda_{\min}(A_m)\right]^T.$$

This $\Lambda_{\min}$ vector captures the smallest eigenvalues for each constraint matrix A_i.

$$\Lambda_{\max} = \left[\lambda_{\max}(A_1) \quad \lambda_{\max}(A_2) \quad \dots \quad \lambda_{\max}(A_m)\right]^T.$$

Similarly, the vector $\Lambda_{\max}$ is defined as the vector of the largest eigenvalues for all constraints.

$$\varrho = \left[\rho(A_1) \quad \rho(A_2) \quad \dots \quad \rho(A_m)\right]^T.$$

This vector captures the ranks for each constraint matrix A_i.

$$\Phi_s = \left[\mathcal{P}_s(A_1) \quad \mathcal{P}_s(A_2) \quad \dots \quad \mathcal{P}_s(A_m)\right]^T.$$

This vector captures the sparsity pattern for each constraint matrix A_i for $s \in \mathcal{S}$.

These vectors provide specific information about all constraints, enabling us to create features that reflect their distributions. To further enhance our feature design, we define several statistical functions.

Notation 7. *We define the following statistics for a given vector x:*

- $\min(x)$*: The minimum value of the elements in the vector x.*
- $\max(x)$*: The maximum value of the elements in the vector x.*

- $\operatorname{avg}(x)$: *The average (mean) value of the elements in the vector x, calculated as:*

$$\operatorname{avg}(x) = \frac{1}{n}\sum_{i=1}^{n} x_i,$$

where n is the number of elements in the vector x.

- $\mu_3(x)$: *Third moment, calculated as $\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^3$, where μ is the mean of x.*

- $\operatorname{IQR}(x)$: *Interquartile Range (IQR), calculated as $Q_3 - Q_1$, where Q_3 is the third quartile and Q_1 is the first quartile of x.*

- $P_{out}(x)$: *Proportion of outliers. Outliers are typically defined based on a specific rule, such as values more than 1.5 times the IQR above the third quartile or below the first quartile.*

- $\operatorname{CV}(x)$: *Coefficient of Variance (CV), calculated as $\frac{\sigma}{\mu}$, where σ is the standard deviation and μ is the mean of x.*

As an example, the function $\min(x)$ will help us determine the minimum of all the minimum eigenvalues of each constraint by evaluating $\min(\Lambda_{\min})$.

Definition 4. *Let $x \in \mathbb{R}^m$ and $h_q(x): \mathbb{R}^m \to \mathbb{R}$ be a function representing the statistic q of x.*

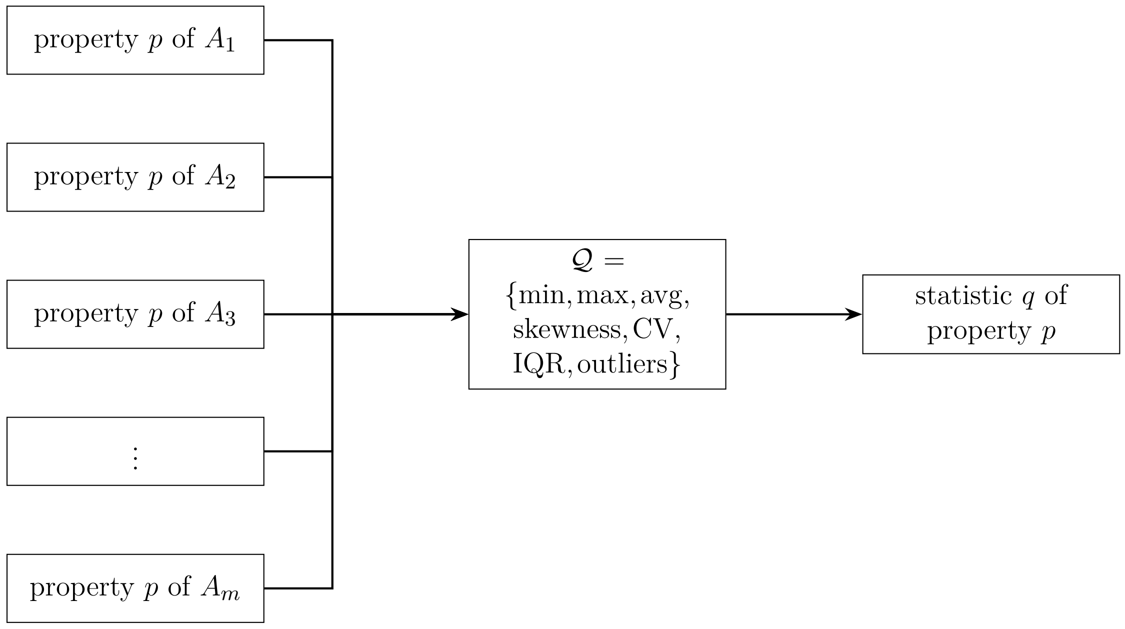
$$h_q(x) = \begin{cases} \min(x) & \textit{if } q = \min \\ \max(x) & \textit{if } q = \max \\ \operatorname{avg}(x) & \textit{if } q = \operatorname{avg} \\ \mu_3(x) & \textit{if } q = \text{thirdmoment} \\ \operatorname{IQR}(x) & \textit{if } q = \text{IQR} \\ P_{out}(x) & \textit{if } q = \textit{outliers} \\ \operatorname{CV}(x) & \textit{if } q = \text{CV}. \end{cases}$$

Figure 3.3 shows how the property p of matrices A_1 to A_m is analyzed through various statistical measures ($\mathcal{Q}$), resulting in a specific statistic q. For example, $h_q(\Theta)$ provides various statistics about the negative eigenvalue ratios of the constraints.

Definition 5. *Let $g_\alpha(x)$ be a function $\mathbb{R}^m \to \mathbb{R}$ defined as follows:*

$$g_\alpha(x) = \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}_{\{x_i=\alpha\}},$$

Figure 3.3 Diagram illustrating the calculation of statistic q for property p of matrices A_1 to A_m.


property p of A_1
property p of A_2
property p of A_3
$\vdots$
property p of A_m
$\mathcal{Q} = \{\min, \max, \text{avg}, \text{skewness}, \text{CV}, \text{IQR}, \text{outliers}\}$
statistic q of property p

where $x \in \mathbb{R}^m$ *and* $\mathbf{1}_{\{x_i=\alpha\}}$ *is the indicator function that equals 1 if* $x_i = \alpha$ *and 0 otherwise.*

This function counts the occurrences of α in the vector x and normalizes the count by dividing by the number of constraints. For example, $g_0(\Theta)$ essentially represents the proportion of convex constraints by counting the zeros in the negative eigenvalue ratios and dividing by m.

Table 3.7 provides an overview of the notations and descriptions for the dimension-independent setup employed in our study. Different from Section 3.3.1.2, the spectral properties section includes statistical measures of vectors representing different eigenvalue characteristics and proportion of convex and concave constraints. In the sparsity pattern section, new ratios are defined to reflect the prevalence of diagonal, hollow, bipartite, tree, chordal and planar matrices among the constraints. As in Sections 3.3.1.1 and 3.3.1.2, this table also includes a feature that indicates whether the original problem has inherent finite variable bounds or if artificial bounds were used. What sets this approach apart is the development of dimension-independent features by offering comprehensive information about all constraints.

Table 3.7 Description of Dimension-Independent Setup.

Notation	Description
Spectral properties	
$\eta(\lambda(A_0))/n$	Number of negative eigenvalues of A_0 over n

$\theta\left(\lambda(A_0)\right)$	Negative eigenvalue ratio of A_0
$\lambda_{\min}\left(A_0\right)$	Smallest eigenvalue of A_0 over n
$\lambda_{\max}\left(A_0\right)$	Largest eigenvalue of A_0 over n
$\rho(A_0)/n$	Rank of A_0 over n
$g_0(\Theta)$	Proportion of convex constraints
$g_1(\Theta)$	Proportion of concave constraints
$h_q(\Theta)$	Statistic q of Θ, $\quad q \in \mathcal{Q}$
$h_q(\Lambda_{\min})$	Statistic q of $\Lambda_{\min}$, $\quad q \in \mathcal{Q}$
$h_q(\Lambda_{\max})$	Statistic q of $\Lambda_{\max}$, $\quad q \in \mathcal{Q}$
$h_q(\varrho)$	Statistic q of ϱ, $\quad q \in \mathcal{Q}$
Sparsity pattern	
$\mathcal{P}_s(A_0)$	1 if A_0 has a sparsity pattern of s $\in \mathcal{S}$, 0 otherwise
$g_s(\Phi_s)$	The ratio of matrices with sparsity pattern of s to the total number of constraints
BoundsExist?	1 if the original problem has finite variable bounds, 0 if artificial bounds method are derived
Total Number of Features	$(\lvert\Pi\rvert \times \lvert\mathcal{Q}\rvert) + 2 \times \lvert\mathcal{S}\rvert + 8 = 55$

Definition 6. *In the context of this calculation,* Π *is a set of properties defined as:*

$$\Pi = \{\Xi, \Theta, \Lambda_{\min}, \Lambda_{\max}, \varrho\}.$$

Table 3.8 presents a detailed comparative analysis of three methodologies for feature construction of QCQP instances with $n = 3$ and $m = 2$. The column labeled *fully_dep* includes features whose number significantly depends on both the dimensionality of the decision variable and the number of constraints (see Section 3.3.1.1). In contrast, the *semi_dep* column encompasses a method where the number of features is independent of n, yet remains dependent on m (see Section 3.3.1.2). Finally, the *dim_indep* column represents a feature extraction approach that is independent of both n and m (see Section 3.3.1.3).

Table 3.8 Example features with $n = 3, m = 2$.

fully_dep	**semi_dep**	**dim_indep**
$\lambda_1(A_0)$	$\theta\,(A_0)$	
$\lambda_2(A_0)$	$\lambda_{\min}\,(A_0)$	
$\lambda_3(A_0)$	$\lambda_{\max}\,(A_0)$	
$\lambda_1(A_1)$	$\theta\,(A_1)$	$h_q(\Theta), \quad q \in \mathcal{Q}$
$\lambda_2(A_1)$	$\lambda_{\min}\,(A_1)$	$g_0(\Theta)$
$\lambda_3(A_1)$	$\lambda_{\max}\,(A_1)$	$g_1(\Theta)$
$\lambda_1(A_2)$	$\theta\,(A_2)$	$h_q(\Lambda_{\min}), \quad q \in \mathcal{Q}$
$\lambda_2(A_2)$	$\lambda_{\min}\,(A_2)$	$h_q(\Lambda_{\max}), \quad q \in \mathcal{Q}$
$\lambda_3(A_2)$	$\lambda_{\max}\,(A_2)$	
$\eta(\lambda(A_0))$	$\eta(\lambda(A_0))/n$	
$\eta(\lambda(A_1))$	$\eta(\lambda(A_1))/n$	$h_q(\mathcal{N}), \quad q \in \mathcal{Q}$
$\eta(\lambda(A_2))$	$\eta(\lambda(A_2))/n$	
$\rho(A_0)$	$\rho(A_0)/n$	
$\rho(A_1)$	$\rho(A_1)/n$	$h_q(\varrho), \quad q \in \mathcal{Q}$
$\rho(A_2)$	$\rho(A_2)/n$	
$\mathcal{P}_D(A_0)$		
$\mathcal{P}_H(A_0)$		
$\mathcal{P}_B(A_0)$		
$\mathcal{P}_T(A_0)$		
$\mathcal{P}_C(A_0)$		
$\mathcal{P}_P(A_0)$		
$\mathcal{P}_D(A_1)$		$g_D(\Phi_D)$
$\mathcal{P}_D(A_2)$		
$\mathcal{P}_H(A_1)$		$g_H(\Phi_H)$
$\mathcal{P}_H(A_2)$		
$\mathcal{P}_B(A_1)$		$g_B(\Phi_B)$
$\mathcal{P}_B(A_2)$		
$\mathcal{P}_T(A_1)$		$g_T(\Phi_T)$
$\mathcal{P}_T(A_2)$		
$\mathcal{P}_C(A_1)$		$g_C(\Phi_C)$
$\mathcal{P}_C(A_2)$		
$\mathcal{P}_P(A_1)$		$g_P(\Phi_P)$
$\mathcal{P}_P(A_2)$		
BoundsExist?		

3.3.2 Data Labeling Workflow

In this section, we present the data labeling workflow used in our study. We create target labels using the optimal values of two different relaxation methods. In the context of our learning experiments, every data point has been labeled in two different ways: binary for classification and continuous for regression. With respect to the binary labels, data points were classified into either LP-favoring or SDP-favoring. In the regression model, continuous labels were applied to data points. These labels were represented as scaled ratios of the differences between the two relaxations. The process for computing these labels is outlined below.

Definition 7. *Let us define the normalized difference between the optimal values of two relaxations as:*

$$\textit{relative difference} = \frac{z_{relaxation1} - z_{relaxation2}}{|z_{relaxation1}| + |z_{relaxation2}|}.$$

The resulting continuous value is used as the label for regression. However, to assign a target label for classification, we compare this relative difference with a predefined threshold:

$$\text{label} = \begin{cases} 0 & \text{if normalized\_diff} < -\text{tolerance} \quad \text{(relaxation2-favoring)} \\ 1 & \text{otherwise} \quad \text{(relaxation1-favoring).} \end{cases}$$

Here, tolerance is an important parameter, which allows for small differences between the lower bounds given by the two relaxations. Setting it to 10^{-2}, we ensure precision in the calculations. Our labeling strategy puts more emphasis on relative performances of the two relaxation methods, avoiding scale differences. Therefore, the tolerance threshold makes the classification flexible; which takes into account the fact that small changes may not actually mean a true preference of one method over another. In this context, the LP relaxation is chosen for relaxation1, while the second relaxation can be either SDP or SDP'. The objective is to establish a time-efficient target by selecting LP when the bounds provided by LP and SDP are approximately equivalent, given that SDP typically requires more runtime.

3.3.3 Training and Testing

Upon completing the data preprocessing and feature engineering processes, raw data is organized and is suitable for machine learning. First, we use the classification model to categorize QCQP instances as LP-favoring or SDP-favoring. In addition, we use a regression model to estimate the difference between the lower bounds offered by these two relaxations. This approach allowed us to understand their comparative effectiveness more comprehensively.

We employed supervised learning techniques for both classification and regression tasks. For the classification tasks, we used RandomForestClassifier (RFC), Support Vector Classifier (SVC), XGBClassifier, GradientBoostingClassifier (GBC), and Neural Network (NN). For the regression tasks, we utilized Random Forest Regressor (RFR), Gradient Boosting Regressor (GBR), Support Vector Regressor (SVR). We tuned those models to find the best performance using a grid search method where we specify a range of values for key hyper-parameters. It permitted spotting the best hyper-parameters combination of every model. The dataset was randomly divided into training and test sets with an 80%–20% ratio. The learning experiments were implemented using Python 3.11.5 and Scikit-learn version 1.3.0. The experiments were executed on a system equipped with an Intel(R) Xeon(R) W-2145 CPU @ 3.70 GHz and 64 GB of RAM.

4. RESULTS

In this chapter, we present the performance evaluations of our learning experiments. First, all experiments conducted in this study are enumerated. A total of 68 experiments were performed for both the LP vs. SDP benchmark and the LP vs. SDP' benchmark, encompassing classification and regression experiments, on 17 datasets that differed in terms of n, m, number of QCQP instances, and feature design approach. Numerous machine learning models, as discussed in Section 3.3.3, were evaluated in these experiments. In the majority of experiments, Gradient Boosting and Random Forest emerged as the top-performing models for both classification and regression. In Section 4.1, we describe the performance metrics and results of the classification models. In Section 4.2, we provide detailed computational results and analysis for the regression models.

Table 4.1 summarizes the experiments carried out in this study, detailing data identifiers, the approaches used for feature engineering, and the size of various datasets used for training and testing.

The columns are explained as follows:

- **Data Identifier**: A unique label for each experiment.
- **Features**: Specifies the feature engineering approach applied, which can be one of the following:
  - *fully_dep*: Problem dimensionality dependent features (see Section 3.3.1.1).
  - *semi_dep*: Variable dimensionality independent features (see Section 3.3.1.2).
  - *dim_indep*: Problem dimensionality independent features (see Section 3.3.1.3).
- **Train** and **Test**: Each represented by three columns: n, m, and *data_size* as the number of QCQP instances.

The experiments vary by the number of decision variables, the number of constraints, and the size of the datasets. For example, the experiment *D-5-1* uses dependent features with $n = 5$, single constraint, and a training dataset of 50,000 instances, using the same configuration for testing.

Special cases are highlighted, such as *S-5,10-2'* and *I-10-2'*, where the training and testing dataset configurations differ. These cases involve a specific training dataset and a differently configured testing dataset, indicated in the Test column.

For more details on the feature engineering approaches (*fully_dep*, *semi_dep*, *dim_indep*), refer to Section 3.3.1.

Table 4.1 List of Experiments.

Data Identifier	**Features**	**Train**			**Test**		
		n	m	**size**	n	m	**size**
D-5-1	fully_dep	5	1	50K	5	1	50K
D-5-2	fully_dep	5	2	50K	5	2	50K
D-10-1	fully_dep	10	1	50K	10	1	50K
D-10-2	fully_dep	10	2	50K	10	2	50K
S-5-1	semi_dep	5	1	50K	5	1	50K
S-5-2	semi_dep	5	2	50K	5	2	50K
S-5-3	semi_dep	5	3	5K	5	3	5K
S-5-10	semi_dep	5	10	10K	5	10	10K
S-10-1	semi_dep	10	1	50K	10	1	50K
S-10-2	semi_dep	10	2	50K	10	2	50K
S-5,10-1	semi_dep	5&10	1	100K	5&10	1	100K
S-5,10-2	semi_dep	5&10	2	100K	5&10	2	100K
S-5,10-2'	semi_dep	5&10	2	100K	20	2	10K
I-5-10	dim_indep	5	10	10K	5	10	10K
I-5-2	dim_indep	5	2	50K	5	2	50K
I-10-2	dim_indep	10	2	50K	10	2	50K
I-10-2'	dim_indep	10	2	50K	5	10	10K

4.1 Classification Model Performance

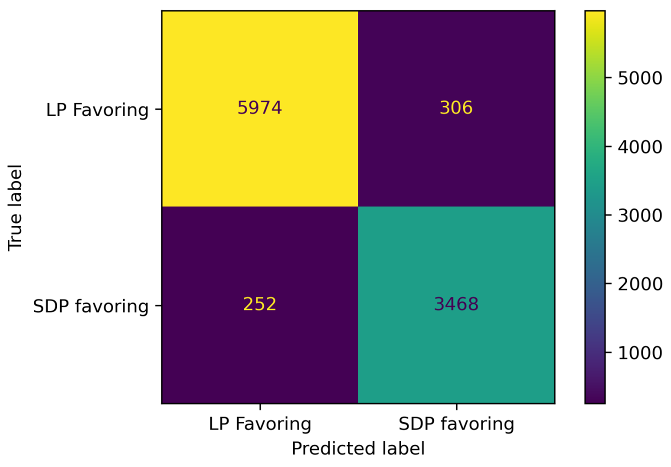
Table 4.2 presents classification performance metrics for different experiments, including accuracy, F1 score, precision, and recall. The general trend is that with increasing n, performance metrics are improving, while the increase in the number of constraints has a negative effect on the success of prediction. The effect of m can be observed from the gradual decrease in the performance of experiments *S-5-1*, *S-5-2*, *S-5-3*, *S-5-10*. The results show that experiments *D-10-1* and *S-10-1* achieved the highest performance across all metrics; this can perhaps be explained by the advantage of dealing with a single-constraint problem. In contrast, experiments *I-10-2'* and *S-5,10-2'* show the lowest performance and show difficulties due to the possible complexity of training and testing on different data. In summary, the results show that changes in problem size and data configuration can significantly affect classification metrics.

Table 4.2 Best Classification Metrics for Different Experiments (LP vs. SDP).

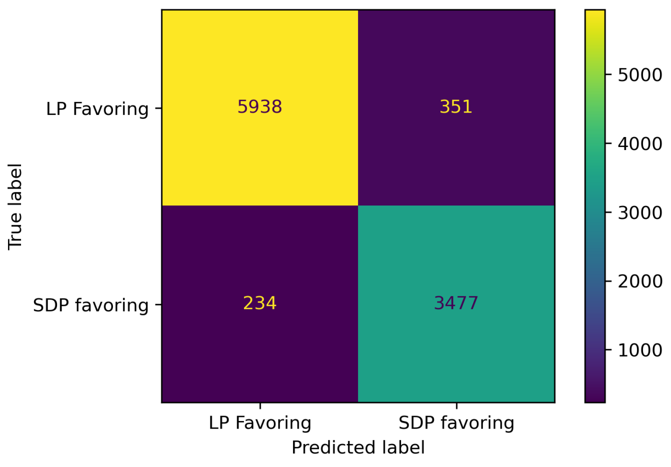
Data Identifier	**Best Model**	**Accuracy**	**F1 Score**	**Precision**	**Recall**
D-5-1	GBC	0.952	0.952	0.953	0.952
D-5-2	GBC	0.935	0.934	0.934	0.935
D-10-1	GBC	0.972	0.972	0.972	0.972
D-10-2	RFC	0.970	0.970	0.970	0.970
S-5-1	GBC	0.950	0.949	0.950	0.950
S-5-2	GBC	0.934	0.934	0.934	0.934
S-5-3	RFC	0.896	0.895	0.896	0.896
S-5-10	GBC	0.846	0.846	0.847	0.846
S-10-1	GBC	0.972	0.972	0.973	0.972
S-10-2	GBC	0.967	0.967	0.967	0.967
S-5,10-1	GBC	0.961	0.961	0.961	0.961
S-5,10-2	GBC	0.949	0.948	0.949	0.949
S-5,10-2'	GBC	0.899	0.901	0.910	0.899
I-5-2	NN	0.933	0.933	0.933	0.933
I-5-10	GBC	0.846	0.846	0.846	0.846
I-10-2	GBC	0.967	0.967	0.967	0.967
I-10-2'	NN	0.807	0.806	0.824	0.807

Figure 4.1 presents confusion matrices for different feature configurations: dimension-dependent (a), semi-dependent setup (b), and fully independent features (c).

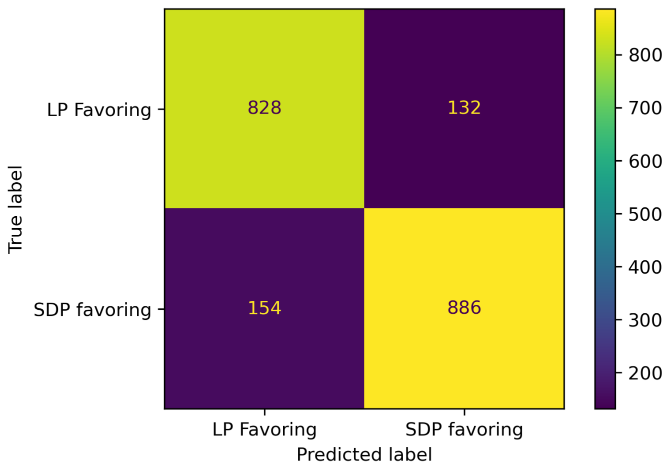
Figure 4.1(a) shows the confusion matrix associated with experiment *D-5-1*. Here, performance is rather balanced between the instances favoring LP and those favoring SDP, with high true positive and true negative rates. Nonetheless, the number of false positives and false negatives indicates an area for improvement in classification accuracy. Figure 4.1(b) demonstrates the confusion matrix of *S-5-1* showing significantly high true positives and true negatives along the diagonal. This indicates that our generalization attempts have been successful. The last figure 4.1(c) presents the confusion matrix for the experiment *I-5-10*, indicating a proportional increase in false positives and false negatives, which may be due to the additional complexity introduced by the feature design. Overall, the high values along the diagonals of these matrices demonstrate the effectiveness of our models, although the dimension-independent setup introduce complexity that may hinder accuracy.



(a) Confusion Matrix for fully_dep ($n = 5$, $m = 1$)



(b) Confusion Matrix for semi_dep ($n = 5$, $m = 1$)



(c) Confusion Matrix for dim_indep ($n = 5$, $m = 10$)

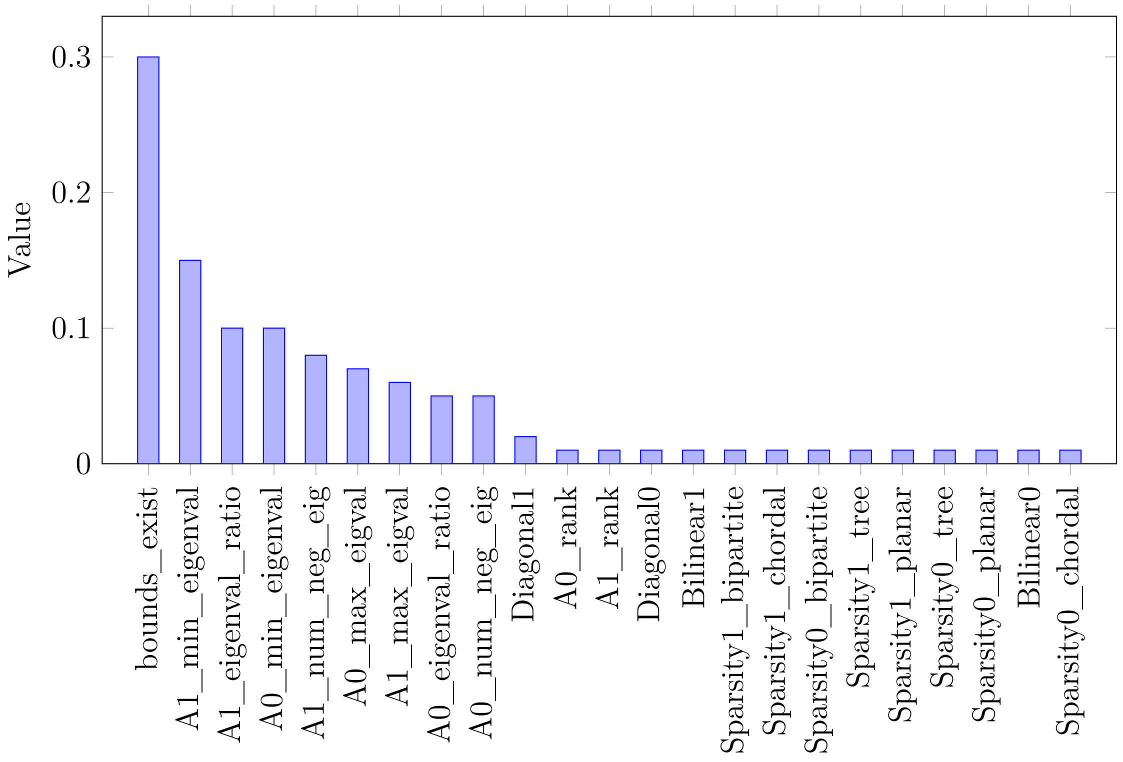
Figure 4.1 Confusion Matrices for Different Feature Designs.

The feature importance chart for *S-5-1* in Figure 4.2 demonstrates that *bounds_exist* is the most critical feature, with importance around 0.30. The classification result is very seriously influenced by the presence of bounds. Also significant are the minimum eigenvalue of matrix A_1 (*A1_min_eigenval*), around 0.15, and the eigenvalue ratio of A_1 (*A1_eigenval_ratio*), around 0.10.

The maximum eigenvalue of A_1 (*A1_max_eigenval*), also important in understanding the convexity of the constraint, has a large influence, with an importance of about 0.06. Other relevant features include the minimum eigenvalue and number of negative eigenvalues of A_0 (*A0_min_eigenval* and *A0_num_neg_eig*), each with importance around 0.10 and 0.05, respectively.

Among the less influential features are ranks and diagonals of matrices and some sparsity properties, with importances near 0.01. We can summarize that the metrics related to eigenvalues and the existence of bounds are the most influential features by a large difference.

Figure 4.2 Feature Importances for semi_dep ($n = 5\&\ m = 1$).



Up to this point, we have meticulously analyzed the classification results of LP versus SDP. We now present the comparative results of LP versus SDP'. The most notable changes in Table 4.3 across the experiments are observed with the increase in dimensions and the complexity of constraints. For fully dependent features, metrics improve significantly when moving from 5 to 10 dimensions, with accuracy and F1 scores rising from around 0.85 in D-5-1 and *D-5-2* to approximately 0.90 in *D-10-1* and *D-10-2*. In semi_dep feature design, *S-5-10*, with ten constraints, shows a drop in performance (accuracy around 0.81) compared to *S-5-1* and *S-5-2* (around 0.84 to 0.85). However, when n is equal to 10, the performance metrics rebound to around 0.90 in *S-10-1* and *S-10-2*. For independent features, the experiments *I-5-2* and *I-10-2* demonstrate surprisingly very high metrics (around 0.93 to 0.96), indicating that higher dimensions and independent features significantly enhance model performance. As expected, *I-10-2'* shows a dip in metrics (around 0.80), highlighting challenges with specific feature and constraint configurations.

Table 4.3 Best Classification Metrics for Different Experiments (LP vs. SDP').

Data Identifier	**Best Model**	**Accuracy**	**F1 Score**	**Precision**	**Recall**
D-5-1	GBC	0.859	0.859	0.859	0.859
D-5-2	GBC	0.853	0.852	0.852	0.853
D-10-1	GBC	0.903	0.902	0.902	0.903
D-10-2	GBC	0.892	0.891	0.890	0.892
S-5-1	GBC	0.846	0.846	0.845	0.846
S-5-2	GBC	0.852	0.851	0.851	0.852
S-5-3	GBC	0.846	0.845	0.844	0.846
S-5-10	GBC	0.816	0.806	0.805	0.816
S-10-1	GBC	0.903	0.902	0.902	0.903
S-10-2	XGB	0.889	0.888	0.887	0.889
S-5,10-1	GBC	0.874	0.873	0.873	0.874
S-5,10-2	GBC	0.875	0.874	0.873	0.875
S-5,10-2'	XGB	0.864	0.845	0.846	0.864
I-5-2	GBC	0.934	0.934	0.934	0.934
I-5-10	GBC	0.846	0.846	0.846	0.846
I-10-2	GBC	0.890	0.889	0.888	0.890
I-10-2'	NN	0.807	0.798	0.796	0.807

4.2 Regression Model Performance

Table 4.4 provides the regression performance metrics for various experiments, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R^2 score, and lastly r_accuracy, which is not a standard regression metric but is specifically designed for the purposes of our analysis.

For a vector of true labels y and a vector of predicted labels $\hat{y}$, we define the performance metric *r_accuracy* as follows:

$$\text{(4.1)}\quad \text{r\_accuracy}(y, \hat{y}) = \frac{1}{K} \sum_{k=1}^{K} \mathbf{1}\left\{\text{sign}(y_k) = \text{sign}(\hat{y}_k)\,\text{or}\;(-\epsilon < y_k < 0\,\text{and}\,\hat{y}_k > 0)\right\}$$

where K is the size of the test set, $\mathbf{1}\{\cdot\}$ is the indicator function, and ϵ is a predefined threshold for near-zero values. Our purpose in developing this performance metric is to consider predictions with the same sign as the actual relative difference as correctly categorized, regardless of their magnitude, and to also accept LP predictions in cases where SDP is tighter than LP by ϵ difference. This metric also provides the advantage of making comparisons with classification accuracy. In this study, ϵ is determined as 10^{-1}.

Table 4.4 Best Regression Metrics for Different Experiments (LP vs. SDP).

Data Identifier	**Best Model**	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
D-5-1	RFR	0.072	0.024	0.155	0.956	0.958
D-5-2	GBR	0.135	0.047	0.216	0.917	0.951
D-10-1	RFR	0.038	0.009	0.096	0.985	0.981
D-10-2	RFR	0.058	0.016	0.126	0.975	0.981
S-5-1	GBR	0.081	0.024	0.156	0.956	0.955
S-5-2	GBR	0.131	0.046	0.215	0.918	0.951
S-5-3	RFR	0.155	0.061	0.246	0.886	0.931
S-5-10	GBR	0.129	0.036	0.190	0.898	0.918
S-10-1	RFR	0.038	0.009	0.095	0.986	0.980
S-10-2	RFR	0.056	0.015	0.124	0.976	0.981
S-5,10-1	RFR	0.056	0.016	0.126	0.973	0.969
S-5,10-2	RFR	0.088	0.031	0.175	0.949	0.965
S-5,10-2’	GBR	0.271	0.126	0.355	0.821	0.981
I-5-2	GBR	0.132	0.047	0.216	0.917	0.952
I-5-10	GBR	0.132	0.038	0.195	0.893	0.916
I-10-2	RFR	0.057	0.016	0.125	0.976	0.981
I-10-2’	GBR	0.386	0.265	0.515	0.215	0.887

Table 4.4 shows significant performance variations between experiments with different configurations. Increasing n from 5 to 10 in the fully dimension dependent setup greatly improves performance, as seen by MAE decreasing from 0.135 to 0.038 and R^2 from 0.917 to 0.985. However, adding more constraints increases errors and reduces R^2, indicating a complexity penalty. For non-independent features, experiments with smaller datasets or more constraints show higher errors and lower R^2, particularly in *S-5-3* and *S-5,10-2*. Independent features generally yield high performance, but complex configurations like *I-10-2'* cause significant drops in metrics, with MAE rising to 0.386 and R^2 dropping to 0.215, demonstrating the challenges of high-dimensional, independent feature setups.

In addition, the *r_accuracy* metric, which tracks how accurately the predicted signs match the actual signs, typically reflects the trends seen in other metrics. Models operating in high-dimensional and less constrained environments tend to show higher *r_accuracy* values. This indicates that the model is more frequently correct in predicting the sign of the output in simpler setups. For example, *r_accuracy* is notably high, around 0.981, in the *D-10-1* and *D-10-2* experiments, consistent with their strong performance in regression tasks. On the other hand, in more complex setups, especially those with independent features and higher dimensions such as *I-10-2'*, there is a noticeable drop in *r_accuracy* to about 0.887. This decrease underscores the model's difficulty in consistently predicting the correct sign under more challenging conditions.

Table 4.5 Best Regression Metrics for Different Experiments (LP vs. SDP').

Data Identifier	**Best Model**	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
D-5-1	RFR	0.065	0.011	0.106	0.885	0.759
D-5-2	GBR	0.074	0.013	0.115	0.888	0.807
D-10-1	RFR	0.058	0.008	0.089	0.930	0.860
D-10-2	GBR	0.067	0.010	0.099	0.921	0.886
S-5-1	GBR	0.063	0.010	0.101	0.896	0.758
S-5-2	GBR	0.074	0.013	0.114	0.890	0.806
S-5-3	GBR	0.077	0.014	0.119	0.888	0.841
S-5-10	GBR	0.064	0.010	0.101	0.937	0.863
S-10-1	GBR	0.060	0.008	0.089	0.930	0.856
S-10-2	RFR	0.067	0.010	0.100	0.919	0.889
S-5,10-1	GBR	0.064	0.010	0.098	0.911	0.800
S-5,10-2	RFR	0.071	0.012	0.109	0.902	0.851
S-5,10-2'	RFR	0.149	0.045	0.213	0.647	0.926
I-5-2	GBR	0.074	0.013	0.114	0.890	0.806
I-5-10	GBR	0.132	0.038	0.195	0.892	0.916
I-10-2	GBR	0.068	0.010	0.100	0.920	0.888
I-10-2'	GBR	0.150	0.040	0.201	0.745	0.842

Having thoroughly examined the regression outcomes of LP compared to SDP, we now shift our focus to the comparative analysis of LP and SDP'. Comparing the tables 4.4 and 4.5, the most drastic changes are seen in the *D-5-1* and *D-10-1* experiments. For *D-5-1*, while MAE decreased from 0.072 to 0.065 and MSE from 0.024 to 0.011, indicating better performance with LP vs. SDP', the R^2 value dropped significantly from 0.956 to 0.885, and r_accuracy from 0.958 to 0.759, highlighting a decrease in overall predictive power despite lower error metrics. In *D-10-1*, MAE increased from 0.038 to 0.058 and R^2 dropped from 0.985 to 0.930, with r_accuracy also decreasing from 0.981 to 0.860, suggesting LP vs. SDP' struggled more with higher dimensions compared to LP vs. SDP. Additionally, in *S-5,10-2'*, the MAE decreased from 0.271 to 0.1493, and MSE from 0.126 to 0.0454, with R^2 improving from 0.821 to 0.6470, but r_accuracy dropping from 0.981 to 0.9262, indicating that while LP vs. SDP' resulted in better error metrics, its predictive consistency was slightly lower in terms of sign accuracy. However, compared to SDP, SDP' relaxation is a much stronger competitor for LP relaxation and can give much tighter bounds than SDP by making good use of variable bounds. For this reason, the high performance of these two relaxations can make it difficult to distinguish significant differences between them. Nevertheless, we observe that it is still successful in predicting the relaxation type, which has superior performance.

5. CONCLUSION

This thesis focuses on analyzing and comparing the performance of two primary relaxations used in QCQP problems. By examining the structural properties of QCQP instances, we formulate conjectures and classify the instances as favoring either SDP or LP relaxations prior to solving them. Our assumptions closely match the data findings, and the machine learning model strongly confirms these results. Three distinct feature generation approaches successfully classify QCQP instances into SDP-favoring or LP-favoring categories. Notably, whether the original problem has finite variable bounds is a primary distinguishing feature. Spectral properties of the data matrices are also critical indicators, ranking as the second most important feature category, followed by other spectral properties.

The developed models provide a significant advantage by predicting the most beneficial relaxation type for a new QCQP instance without requiring users to test various relaxation methods. This prediction capability is independent of the instance size, offering an efficient approach to selecting relaxation techniques, thereby saving time and computational resources.

A significant constraint that arose and was particularly noticeable during this research was the prolonged execution time for regression, especially when the number of features is substantial. This issue highlights the need for optimization in the feature selection and model training process.

Future study should focus on these several promising directions. Firstly, incorporating the relaxation runtimes into the prediction model may offer a more comprehensive evaluation of the efficiency of the relaxation techniques. Secondly, the practicality of the model could be improved by investigating the use of alternative features that are easier to get, including eigenvalue estimates rather than precise calculations. Third, the accuracy of the classification could be improved by looking into various sparsity patterns for the set $\mathcal{S}$ and a different set of statistics $\mathcal{Q}$ that go along with it. The final goal is to confirm the model's efficacy in real-world circumstances by testing these concepts on MINLPLib instances (Koch, Berthold, Pfetsch

& Wolter, 2021). This will offer a thorough assessment of the model's capacity to predict outcomes and its usefulness in a variety of QCQP scenarios. As a result, this thesis opens the door for more focused and successful relaxation method selection in subsequent studies and applications.

BIBLIOGRAPHY

Anstreicher, K. (2009). Semidefinite programming versus the reformulation-linearization technique for nonconvex quadratically constrained quadratic programming. *Journal of Global Optimization*, *43*, 471–484.

Bai, X., Wei, H., Fujisawa, K., & Wang, Y. (2008). Semidefinite programming for optimal power flow problems. *International Journal of Electrical Power & Energy Systems*, *30*(6-7), 383–392.

Beck, A. & Eldar, Y. C. (2003). Strong duality in nonconvex quadratic optimization with two quadratic constraints. *SIAM Journal on Optimization*, *17*(3), 844–860.

Bonami, P., Lodi, A., & Zarpellon, G. (2017). Learning a classification of mixed-integer quadratic programming problems. In *Integration of AI and OR Techniques in Constraint Programming.*

Bonami, P., Lodi, A., & Zarpellon, G. (2022). A classifier to decide on the linearization of mixed-integer quadratic problems in cplex. *Operations Research*, *70*(6), 3303–3320.

Burer, S. & Ye, Y. (2018). Exact semidefinite formulations for a class of (random and non-random) nonconvex quadratic programs.

Diamond, S. & Boyd, S. (2016). CVXPY: A Python-embedded modeling language for convex optimization. Version 1.1.

Fu, M., Luo, Z.-Q., & Ye, Y. (1998). Approximation algorithms for quadratic programming. *Journal of Combinatorial Optimization*, *2*, 29–50.

Gaar, E. & Rendl, F. (2020). A computational study of exact subgraph based sdp bounds for max-cut, stable set and coloring. *Mathematical Programming*, *183*, 283–308.

Ghaddar, B., Gómez-Casares, I., González-Díaz, J., González-Rodríguez, B., Pateiro-López, B., & Rodríguez-Ballesteros, S. (2022). Learning for spatial branching: An algorithm selection approach. Technical report, Technical report.

Goemans, M. X. & Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, *42*(6), 1115–1145.

González-Rodríguez, B., Alvite-Pazó, R., Alvite-Pazó, S., Ghaddar, B., & Díaz, J. G. (2022). Polynomial optimization: Enhancing rlt relaxations with conic constraints. Available at: https://arxiv.org/abs/2208.05608.

Gurobi Optimization, L. (2021). Gurobi optimizer reference manual.

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Networkx. Software available at https://networkx.github.io/.

Hillestad, R. & Jacobsen, S. (1980). Linear programs with an additional reverse convex constraint. *Applied Mathematics and Optimization*, *6*, 257–269.

IBM Corp. (2022). *IBM ILOG CPLEX Optimization Studio. CPLEX User's Manual.* Available at: `https://www.ibm.com/es-es/products/ilog-cplex-optimization-studio`.

Khajavirad, A. (2024). The circle packing problem: a theoretical comparison of various convexification techniques. *arXiv*, (2404.03091). Subjects: Optimization and Control (math.OC).

Kim, S. & Kojima, M. (2003). Exact solutions of some nonconvex quadratic optimization problems via sdp and socp relaxations. *Computational Optimization and Applications*, *26*, 143–154.

Koch, T., Berthold, T., Pfetsch, M. E., & Wolter, K. (2021). MINLPLib - a library of mixed-integer and continuous nonlinear programming instances. Accessed: 2024-07-09.

Kocuk, B., Dey, S. S., & Sun, X. A. (2016). Strong socp relaxations for the optimal power flow problem. *Operations Research*, *64*(6), 1177–1196.

Kruber, M., Lübbecke, M., & Parmentier, A. (2017). Learning when to use a decomposition. (pp. 202–210).

Luo, Z.-Q., Ma, W.-K., So, A. M.-C., Ye, Y., & Zhang, S. (2010). Semidefinite relaxation of quadratic optimization problems. *IEEE Signal Processing Magazine*, *27*(3), 20–34.

Marandi, A., Dahl, J., & de Klerk, E. (2018). A numerical evaluation of the bounded degree sum-of-squares hierarchy of lasserre, toh, and yang on the pooling problem. *Annals of Operations Research*, *265*(1), 67–92.

McCormick, G. (1976). Computability of global solutions to factorable nonconvex programs: Part i — convex underestimating problems. *Mathematical Programming*, *10*, 147–175.

MOSEK ApS (2020). The mosek optimization toolbox for matlab manual. version 9.3. Available at http://docs.mosek.com/9.3/toolbox/index.html.

Nemirovskii, A., Roos, C., & Terlaky, T. (1999). On maximization of quadratic form over intersection of ellipsoids with common centers. *Mathematical Programming*, *86*, 463–473.

Nesterov, Y. E. (2000). Global quadratic optimization via conic relaxation. In H. Wolkowicz, R. Saigal, & L. Vandenberghe (Eds.), *Handbook of Semidefinite Programming, Theory, Algorithms, and Applications* (pp. 363–387). Norwell, MA: Kluwer Academic Publishers.

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

Qiu, Y. & Yıldırım, E. A. (2023). Polyhedral properties of rlt relaxations of nonconvex quadratic programs and their implications on exact relaxations.

Torres, F. (1990). Linearization of mixed-integer products. *Mathematical Programming*, *49*, 427–428.

Vandenberghe, L. & Boyd, S. (1996). Semidefinite programming. *SIAM Review*, *38*, 49–95.

Wang, A. L. & Kılınç-Karzan, F. (2022). On the tightness of sdp relaxations of qcqps. *Mathematical Programming*, *193*, 33–73.

Weiner, J., Ernst, A. T., Li, X., & Sun, Y. (2023). Ranking constraint relaxations for mixed integer programs using a machine learning approach. *EURO Journal on Computational Optimization*, *11*, 100061.

Ye, Y. & Zhang, S. (2003). New results on quadratic minimization. *SIAM Journal on Optimization*, *14*(1), 245–267.

Zhang, S. (2000). Quadratic maximization and semidefinite relaxation. *Mathemat-*

ical Programming, Series B, *87*(3), 453–465.

APPENDIX A

Classification Metrics for LP vs. SDP

Table A.1 Classification Metrics for D-5-1 (LP vs. SDP)

Model	Accuracy	F1 Score	Precision	Recall
RandomForestClassifier	0.949	0.949	0.949	0.949
SVC	0.861	0.859	0.860	0.861
XGBClassifier	0.949	0.948	0.948	0.949
GradientBoostingClassifier	0.952	0.952	0.953	0.952
NeuralNetwork	0.745	0.698	0.793	0.745

Table A.2 Classification Metrics for D-5-2 (LP vs. SDP)

Model	Accuracy	F1 Score	Precision	Recall
RandomForestClassifier	0.932	0.931	0.931	0.932
SVC	0.831	0.829	0.830	0.831
XGBClassifier	0.931	0.930	0.930	0.931
GradientBoostingClassifier	0.935	0.934	0.934	0.935
NeuralNetwork	0.865	0.866	0.867	0.865

Table A.3 Classification Metrics for D-10-1 (LP vs. SDP)

Model	Accuracy	F1 Score	Precision	Recall
RandomForestClassifier	0.971	0.971	0.971	0.971
SVC	0.867	0.865	0.865	0.867
XGBClassifier	0.969	0.969	0.969	0.969
GradientBoostingClassifier	0.972	0.972	0.972	0.972
NeuralNetwork	0.916	0.917	0.922	0.916

Table A.4 Classification Metrics for D-10-2 (LP vs. SDP)

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.970	0.970	0.970	0.970
SVC	0.855	0.853	0.854	0.855
XGBClassifier	0.969	0.969	0.969	0.969
GradientBoostingClassifier	0.967	0.967	0.967	0.967
NeuralNetwork	0.740	0.688	0.803	0.740

Table A.5 Classification Metrics for N-5-1 (LP vs. SDP)

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.945	0.945	0.945	0.945
SVC	0.856	0.854	0.855	0.856
XGBClassifier	0.949	0.949	0.949	0.949
GradientBoostingClassifier	0.950	0.949	0.950	0.950
NeuralNetwork	0.913	0.912	0.914	0.913

Table A.6 Classification Metrics for N-5-2 (LP vs. SDP)

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.929	0.929	0.929	0.929
SVC	0.831	0.829	0.830	0.831
XGBClassifier	0.928	0.928	0.928	0.928
GradientBoostingClassifier	0.934	0.934	0.934	0.934
NeuralNetwork	0.881	0.879	0.882	0.881

Table A.7 Classification Metrics for N-5-3 (LP vs. SDP)

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.896	0.895	0.896	0.896
SVC	0.760	0.758	0.758	0.760
XGBClassifier	0.882	0.881	0.882	0.882
GradientBoostingClassifier	0.885	0.884	0.886	0.885
NeuralNetwork	0.865	0.862	0.870	0.865

Table A.8 Classification Metrics for N-5-10 (LP vs. SDP)

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.838	0.838	0.839	0.838
SVC	0.687	0.686	0.693	0.687
XGBClassifier	0.829	0.829	0.829	0.829
GradientBoostingClassifier	0.846	0.846	0.847	0.846
NeuralNetwork	0.701	0.685	0.768	0.701

Table A.9 Classification Metrics for N-10-1 (LP vs. SDP)

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.969	0.969	0.970	0.969
SVC	0.867	0.866	0.866	0.867
XGBClassifier	0.970	0.970	0.970	0.970
GradientBoostingClassifier	0.972	0.972	0.973	0.972
NeuralNetwork	0.928	0.926	0.930	0.928

Table A.10 Classification Metrics for N-10-2 (LP vs. SDP)

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.967	0.967	0.967	0.967
SVC	0.854	0.852	0.853	0.854
XGBClassifier	0.965	0.965	0.965	0.965
GradientBoostingClassifier	0.967	0.967	0.967	0.967
NeuralNetwork	0.675	0.571	0.748	0.675

Table A.11 Classification Metrics for N-5,10-1 (LP vs. SDP)

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.958	0.958	0.958	0.958
SVC	0.866	0.863	0.865	0.866
XGBClassifier	0.960	0.960	0.960	0.960
GradientBoostingClassifier	0.961	0.961	0.961	0.961
NeuralNetwork	0.783	0.752	0.824	0.783

Table A.12 Classification Metrics for N-5,10-2 (LP vs. SDP)

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.949	0.948	0.948	0.949
SVC	0.850	0.847	0.849	0.850
XGBClassifier	0.948	0.947	0.947	0.948
GradientBoostingClassifier	0.949	0.948	0.949	0.949
NeuralNetwork	0.864	0.866	0.870	0.864

Table A.13 Classification Metrics for N-5,10-2' (LP vs. SDP)

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.907	0.908	0.910	0.907
SVC	0.652	0.515	0.425	0.652
XGBClassifier	0.852	0.855	0.876	0.852
GradientBoostingClassifier	0.899	0.901	0.910	0.899
NeuralNetwork	0.731	0.737	0.796	0.731

Table A.14 Classification Metrics for I-5-2 (LP vs. SDP)

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.928	0.928	0.928	0.928
SVC	0.633	0.490	0.400	0.633
XGBClassifier	0.931	0.931	0.931	0.931
GradientBoostingClassifier	0.934	0.934	0.934	0.934
NeuralNetwork	0.933	0.933	0.933	0.933

Table A.15 Classification Metrics for I-5-10 (LP vs. SDP)

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.837	0.837	0.838	0.837
SVC	0.521	0.358	0.751	0.521
XGBClassifier	0.831	0.831	0.831	0.831
GradientBoostingClassifier	0.846	0.846	0.846	0.846
NeuralNetwork	0.836	0.836	0.839	0.836

Table A.16 Classification Metrics for I-10-2 (LP vs. SDP)

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.967	0.967	0.967	0.967
SVC	0.650	0.513	0.423	0.650
XGBClassifier	0.967	0.966	0.966	0.967
GradientBoostingClassifier	0.967	0.967	0.967	0.967
NeuralNetwork	0.964	0.963	0.963	0.964

Table A.17 Classification Metrics for I-10-2' (LP vs. SDP)

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.763	0.758	0.798	0.763
SVC	0.479	0.311	0.230	0.479
XGBClassifier	0.799	0.797	0.822	0.799
GradientBoostingClassifier	0.799	0.798	0.815	0.799
NeuralNetwork	0.807	0.806	0.824	0.807

Classification Metrics for LP vs. SDP'

Table A.18 Classification Metrics for D-5-1 (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.851	0.850	0.850	0.851
SVC	0.636	0.560	0.607	0.636
XGBClassifier	0.853	0.853	0.853	0.853
GradientBoostingClassifier	0.859	0.859	0.859	0.859
NeuralNetwork	0.556	0.526	0.754	0.556

Table A.19 Classification Metrics for D-5-2 (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.845	0.843	0.843	0.845
SVC	0.668	0.535	0.447	0.668
XGBClassifier	0.849	0.848	0.848	0.849
GradientBoostingClassifier	0.853	0.852	0.852	0.853
NeuralNetwork	0.739	0.683	0.784	0.739

Table A.20 Classification Metrics for D-10-1 (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.894	0.892	0.892	0.894
SVC	0.707	0.586	0.500	0.707
XGBClassifier	0.900	0.898	0.898	0.900
GradientBoostingClassifier	0.903	0.902	0.902	0.903
NeuralNetwork	0.770	0.780	0.838	0.770

Table A.21 Classification Metrics for D-10-2 (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.888	0.886	0.885	0.888
SVC	0.761	0.657	0.579	0.761
XGBClassifier	0.886	0.885	0.884	0.886
GradientBoostingClassifier	0.892	0.891	0.890	0.892
NeuralNetwork	0.621	0.644	0.834	0.621

Table A.22 Classification Metrics for N-5-1 (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.834	0.833	0.833	0.834
SVC	0.626	0.547	0.582	0.626
XGBClassifier	0.841	0.841	0.840	0.841
GradientBoostingClassifier	0.846	0.846	0.845	0.846
NeuralNetwork	0.681	0.601	0.749	0.681

Table A.23 Classification Metrics for N52 (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.842	0.839	0.839	0.842
SVC	0.668	0.535	0.446	0.668
XGBClassifier	0.847	0.845	0.845	0.847
GradientBoostingClassifier	0.852	0.851	0.851	0.852
NeuralNetwork	0.752	0.706	0.789	0.752

Table A.24 Classification Metrics for N-5-3 (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.835	0.829	0.829	0.835
SVC	0.721	0.604	0.520	0.721
XGBClassifier	0.828	0.826	0.824	0.828
GradientBoostingClassifier	0.846	0.845	0.844	0.846
NeuralNetwork	0.779	0.726	0.812	0.779

Table A.25 Classification Metrics for N-5-10 (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.808	0.788	0.795	0.808
SVC	0.747	0.638	0.557	0.747
XGBClassifier	0.806	0.797	0.795	0.806
GradientBoostingClassifier	0.816	0.806	0.805	0.816
NeuralNetwork	0.807	0.798	0.796	0.807

Table A.26 Classification Metrics for N-10-1 (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.892	0.891	0.891	0.892
SVC	0.707	0.586	0.500	0.707
XGBClassifier	0.899	0.897	0.897	0.899
GradientBoostingClassifier	0.903	0.902	0.902	0.903
NeuralNetwork	0.823	0.829	0.851	0.823

Table A.27 Classification Metrics for N-10-2 (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.885	0.883	0.882	0.885
SVC	0.761	0.657	0.579	0.761
XGBClassifier	0.889	0.888	0.887	0.889
GradientBoostingClassifier	0.890	0.886	0.887	0.890
NeuralNetwork	0.817	0.776	0.832	0.817

Table A.28 Classification Metrics for N-5,10-1 (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.865	0.864	0.864	0.865
SVC	0.666	0.532	0.443	0.665
XGBClassifier	0.873	0.872	0.872	0.873
GradientBoostingClassifier	0.874	0.873	0.873	0.8742
NeuralNetwork	0.849	0.848	0.847	0.849

Table A.29 Classification Metrics for N-5,10-2 (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.869	0.867	0.866	0.869
SVC	0.715	0.596	0.511	0.715
XGBClassifier	0.873	0.871	0.871	0.873
GradientBoostingClassifier	0.875	0.874	0.873	0.875
NeuralNetwork	0.793	0.754	0.813	0.793

Table A.30 Classification Metrics for N-5,10-2' (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.861	0.831	0.842	0.861
SVC	0.839	0.766	0.704	0.839
XGBClassifier	0.864	0.845	0.846	0.864
GradientBoostingClassifier	0.858	0.842	0.839	0.858
NeuralNetwork	0.823	0.821	0.819	0.823

Table A.31 Classification Metrics for I-5-2 (LP vs.SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.680	0.676	0.681	0.680
SVC	0.550	0.540	0.544	0.550
XGBClassifier	0.710	0.709	0.709	0.710
GradientBoostingClassifier	0.790	0.790	0.792	0.790
NeuralNetwork	0.740	0.739	0.740	0.740

Table A.32 Classification Metrics for I-5-10 (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.845	0.845	0.845	0.845
SVC	0.521	0.358	0.751	0.521
XGBClassifier	0.831	0.831	0.831	0.831
GradientBoostingClassifier	0.846	0.846	0.846	0.846
NeuralNetwork	0.836	0.836	0.839	0.836

Table A.33 Classification Metrics for I-10-2 (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.885	0.883	0.882	0.885
SVC	0.762	0.667	0.721	0.762
XGBClassifier	0.886	0.884	0.884	0.886
GradientBoostingClassifier	0.890	0.889	0.888	0.890
NeuralNetwork	0.884	0.883	0.882	0.884

Table A.34 Classification Metrics for I-10-2' (LP vs. SDP')

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
RandomForestClassifier	0.799	0.795	0.793	0.799
SVC	0.753	0.648	0.783	0.753
XGBClassifier	0.779	0.784	0.793	0.779
GradientBoostingClassifier	0.804	0.796	0.793	0.804
NeuralNetwork	0.807	0.798	0.796	0.807

Regression Metrics for LP vs. SDP

Table A.35 Regression Metrics for D-5-1 (LP vs. SDP)

Model	MAE	MSE	RMSE	R^2	r_accuracy
SVR	0.652	0.497	0.705	0.087	0.698
RandomForestRegressor	0.072	0.024	0.155	0.956	0.958
GradientBoostingRegressor	0.082	0.025	0.159	0.954	0.961

Table A.36 Regression Metrics for D-5-2 (LP vs. SDP)

Model	MAE	MSE	RMSE	R^2	r_accuracy
SVR	0.680	0.560	0.749	0.007	0.671
RandomForestRegressor	0.121	0.047	0.216	0.918	0.948
GradientBoostingRegressor	0.135	0.047	0.216	0.917	0.951

Table A.37 Regression Metrics for D-10-1 (LP vs. SDP)

Model	MAE	MSE	RMSE	R^2	r_accuracy
SVR	0.715	0.589	0.767	0.057	0.709
RandomForestRegressor	0.038	0.009	0.096	0.985	0.981
GradientBoostingRegressor	0.045	0.010	0.098	0.985	0.979

Table A.38 Regression Metrics D-10-2 (LP vs. SDP)

Model	MAE	MSE	RMSE	R^2	r_accuracy
SVR	0.744	0.641	0.801	0.001	0.688
RandomForestRegressor	0.058	0.016	0.126	0.975	0.981
GradientBoostingRegressor	0.067	0.017	0.131	0.973	0.978

Table A.39 Regression Metrics for N-5-1 (LP vs. SDP)

Model	MAE	MSE	RMSE	R^2	r_accuracy
SVR	0.444	0.257	0.507	0.530	0.894
RandomForestRegressor	0.074	0.025	0.157	0.955	0.955
GradientBoostingRegressor	0.081	0.024	0.156	0.956	0.955

Table A.40 Regression Metrics for N-5-2 (LP vs. SDP)

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r__accuracy**
SVR	0.535	0.349	0.591	0.382	0.845
RandomForestRegressor	0.119	0.046	0.215	0.918	0.948
GradientBoostingRegressor	0.131	0.046	0.215	0.918	0.951

Table A.41 Regression Metrics for N-5-3 (LP vs. SDP)

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r__accuracy**
SVR	0.559	0.391	0.625	0.262	0.806
RandomForestRegressor	0.155	0.061	0.246	0.886	0.931
GradientBoostingRegressor	0.167	0.062	0.250	0.882	0.927

Table A.42 Regression Metrics for N-5-10 (LP vs. SDP)

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r__accuracy**
SVR	0.472	0.341	0.584	0.034	0.570
RandomForestRegressor	0.124	0.035	0.188	0.900	0.911
GradientBoostingRegressor	0.129	0.036	0.190	0.898	0.918

Table A.43 Regression Metrics for N-10-1 (LP vs. SDP)

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r__accuracy**
SVR	0.405	0.246	0.496	0.606	0.920
RandomForestRegressor	0.038	0.009	0.095	0.986	0.980
GradientBoostingRegressor	0.044	0.010	0.099	0.984	0.978

Table A.44 Regression Metrics for N-10-2 (LP vs. SDP)

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r__accuracy**
SVR	0.482	0.306	0.553	0.523	0.894
RandomForestRegressor	0.056	0.015	0.124	0.976	0.981
GradientBoostingRegressor	0.066	0.017	0.130	0.974	0.976

Table A.45 Regression Metrics for N-5,10-1 (LP vs. SDP)

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	0.357	0.201	0.448	0.658	0.913
RandomForestRegressor	0.056	0.016	0.126	0.973	0.969
GradientBoostingRegressor	0.064	0.017	0.129	0.971	0.967

Table A.46 Regression Metrics for N-5,10-2 (LP vs. SDP)

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	0.437	0.265	0.514	0.563	0.889
RandomForestRegressor	0.088	0.031	0.175	0.949	0.965
GradientBoostingRegressor	0.104	0.034	0.184	0.944	0.963

Table A.47 Regression Metrics for N-5,10-2' (LP vs. SDP)

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	0.712	0.665	0.815	0.057	0.662
RandomForestRegressor	0.296	0.148	0.385	0.790	0.981
GradientBoostingRegressor	0.271	0.126	0.355	0.821	0.981

Table A.48 Regression Metrics for I-5-2 (LP vs. SDP)

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	2.540	728.711	26.995	-1290.870	0.933
RandomForestRegressor	0.119	0.046	0.213	0.919	0.950
GradientBoostingRegressor	0.132	0.047	0.216	0.917	0.952

Table A.49 Regression Metrics for I-5-10 (LP vs. SDP)

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	0.273	0.142	0.377	0.598	0.797
RandomForestRegressor	0.129	0.038	0.195	0.892	0.914
GradientBoostingRegressor	0.132	0.038	0.195	0.893	0.916

Table A.50 Regression Metrics for I-10-2 (LP vs. SDP)

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	1.837	1349.157	36.731	-2099.320	0.975
RandomForestRegressor	0.057	0.016	0.125	0.976	0.981
GradientBoostingRegressor	0.067	0.017	0.131	0.973	0.976

Table A.51 Regression Metrics for I-10-2' (LP vs. SDP)

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	0.471	1.101	1.049	-2.258	0.721
RandomForestRegressor	0.372	0.254	0.504	0.249	0.878
GradientBoostingRegressor	0.386	0.265	0.515	0.215	0.887

Regression Metrics for LP vs. SDP'

Table A.52 Regression Metrics for D-5-1 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	0.256	0.091	0.302	0.064	0.706
RandomForestRegressor	0.065	0.011	0.106	0.885	0.759
GradientBoostingRegressor	0.066	0.011	0.105	0.886	0.746

Table A.53 Regression Metrics for D-5-2 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	0.301	0.119	0.346	-0.003	0.758
RandomForestRegressor	0.074	0.014	0.117	0.886	0.808
GradientBoostingRegressor	0.074	0.013	0.115	0.888	0.807

Table A.54 Regression Metrics for D-10-1 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	0.291	0.109	0.330	0.043	0.799
RandomForestRegressor	0.058	0.008	0.089	0.930	0.860
GradientBoostingRegressor	0.060	0.008	0.089	0.930	0.854

Table A.55 Regression Metrics for D-10-2 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	0.314	0.125	0.354	-0.006	0.841
RandomForestRegressor	0.067	0.010	0.100	0.920	0.888
GradientBoostingRegressor	0.067	0.010	0.099	0.921	0.886

Table A.56 Regression Metrics for N-5-1 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	0.176	0.046	0.213	0.540	0.728
RandomForestRegressor	0.064	0.011	0.104	0.890	0.759
GradientBoostingRegressor	0.063	0.010	0.101	0.896	0.758

Table A.57 Regression Metrics for N-5-2 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	0.223	0.069	0.262	0.423	0.758
RandomForestRegressor	0.074	0.014	0.117	0.885	0.808
GradientBoostingRegressor	0.074	0.013	0.114	0.890	0.806

Table A.58 Regression Metrics for N-5-3 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	0.252	0.085	0.292	0.328	0.810
RandomForestRegressor	0.077	0.015	0.123	0.880	0.842
GradientBoostingRegressor	0.077	0.014	0.119	0.888	0.841

Table A.59 Regression Metrics for N-5-10 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r__accuracy**
SVR	0.352	0.155	0.393	0.043	0.828
RandomForestRegressor	0.064	0.010	0.101	0.937	0.867
GradientBoostingRegressor	0.064	0.010	0.101	0.937	0.863

Table A.60 Regression Metrics for N-10-1 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r__accuracy**
SVR	0.173	0.045	0.212	0.604	0.799
RandomForestRegressor	0.059	0.008	0.091	0.928	0.861
GradientBoostingRegressor	0.060	0.008	0.089	0.930	0.856

Table A.61 Regression Metrics for N-10-2 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r__accuracy**
SVR	0.202	0.058	0.240	0.536	0.841
RandomForestRegressor	0.067	0.010	0.100	0.919	0.889
GradientBoostingRegressor	0.068	0.010	0.100	0.920	0.885

Table A.62 Regression Metrics for N-5,10-1 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r__accuracy**
SVR	0.154	0.037	0.193	0.654	0.760
RandomForestRegressor	0.062	0.010	0.098	0.910	0.804
GradientBoostingRegressor	0.064	0.010	0.098	0.911	0.800

Table A.63 Regression Metrics for N-5,10-2 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r__accuracy**
SVR	0.184	0.050	0.224	0.586	0.801
RandomForestRegressor	0.071	0.012	0.109	0.902	0.851
GradientBoostingRegressor	0.073	0.012	0.109	0.902	0.844

Table A.64 Regression Metrics for N-5,10-2' (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	0.285	0.135	0.367	-0.049	0.923
RandomForestRegressor	0.149	0.045	0.213	0.647	0.926
GradientBoostingRegressor	0.1478	0.0472	0.2174	0.6324	0.9259

Table A.65 Regression Metrics for I-5-2 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	1.387	276.174	16.618	-2317.090	0.799
RandomForestRegressor	0.074	0.014	0.118	0.884	0.809
GradientBoostingRegressor	0.074	0.013	0.114	0.890	0.806

Table A.66 Regression Metrics for I-5-10 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	0.273	0.142	0.377	0.598	0.797
RandomForestRegressor	0.129	0.038	0.195	0.892	0.914
GradientBoostingRegressor	0.132	0.038	0.195	0.892	0.916

Table A.67 Regression Metrics for I-10-2 (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	1.458	961.248	31.004	-7721.770	0.878
RandomForestRegressor	0.067	0.010	0.101	0.918	0.888
GradientBoostingRegressor	0.068	0.010	0.100	0.920	0.888

Table A.68 Regression Metrics for I-10-2' (LP vs. SDP')

Model	**MAE**	**MSE**	**RMSE**	**R^2**	**r_accuracy**
SVR	3.662	20.380	4.514	-127.379	0.474
RandomForestRegressor	0.165	0.049	0.222	0.689	0.842
GradientBoostingRegressor	0.150	0.040	0.201	0.745	0.842

Counterexample

This counterexample presents a QCQP instance that challenges Conjecture 1. It demonstrates that although SDP relaxation typically provides a tighter bound than LP relaxation with high probability, this condition does not universally hold when artificial bounds are applied.

$$A_0 = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad A_1 = \begin{pmatrix} 1.61 & 1.18 & -0.36 & 1.47 & -1.04 \\ 1.18 & 1.18 & -0.31 & 1.29 & -0.98 \\ -0.36 & -0.31 & 0.21 & -0.41 & 0.53 \\ 1.47 & 1.29 & -0.41 & 1.56 & -1.25 \\ -1.04 & -0.98 & 0.53 & -1.25 & 1.49 \end{pmatrix}$$

$$b_0^T = \begin{pmatrix} 4.89 & 3.85 & -4.09 & 1.78 & -3.28 \end{pmatrix}$$

$$b_1^T = \begin{pmatrix} -0.40 & -4.49 & 2.30 & 2.91 & 2.97 \end{pmatrix}$$

$$c_0 = -7, \quad c_1 = -8$$

$$l^T = \begin{pmatrix} -50.12 & -10.49 & -436.03 & -326.22 & -144.39 \end{pmatrix}$$

$$u^T = \begin{pmatrix} 132.51 & 222.56 & 231.09 & 28.34 & 161.36 \end{pmatrix}$$