Final Year Project

Learning to Sparsify the Capacitated Vehicle Routing Problem Instances

Braddy Yeoh

Student ID: 17357376

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Dr. Deepak Ajwani

UCD School of Computer Science
University College Dublin

May 31, 2023

Table of Contents

Abstract

CVRP is traditionally solved by using exact algorithms or heuristics [1]. Exact algorithms require long running times but provide optimal solutions [1]. Heuristics are quick, but do not guarantee optimality and requires specialist knowledge [1]. The goal of this project is to prune edges of the CVRP that are not part of the optimal solution to decrease the running times while maintaining optimality for exact algorithms. This project aims to leverage classification as it is efficient, scalable, and does not lose optimality. As the problem is NP-hard, any reduction in the problem size will result in evident improvements in the run time given a large enough problem. Classification models learn through features that capture the essence of whether an edge is part of the optimal solution. The best performing model then classifies the edges of unseen CVRP instances to prune away edges that are not part of the solution. The results showcased the best performing model, the random forest, is able to generalise and prune at least half of the edges of unseen CVRP instances, decrease the running time, and maintain optimality. Furthermore, for certain instances, it reduced the running time from 300 seconds to less than 10 seconds after pruning while maintaining optimality. It demonstrates the proposed method in this project is efficient, scalable, does not lose optimality, and reduces the running time of exact algorithms.

Details

- GitLab code repository
- DIMACS - CVRP instances

Chapter 1: Project Specification

In the Capacitated Vehicle Routing Problem (CVRP), the goal is to find the optimal set of routes for a (homogeneous) group of vehicles to traverse to deliver to a set of customers where there is a limited vehicle capacity [2–4].

Popular methods use deep neural networks or reinforcement learning, however, this project opts for a different approach [5]. This approach is similar to the framework presented by Ajwani et al. [6] where it was tested on the maximum-clique problem. The idea of the approach is as follows. Given an arbitrary edge, the machine learning model classifies such an edge with either a positive or negative label. If such a label is positive, then it indicates that the edge is part of the optimal solution. In other words, the edge belongs to the optimal route.

The dataset is presented by DIMACS (http://dimacs.rutgers.edu/programs/challenge/vrp/cvrp/) where the optimal routes are also provided. It is a collection of standard benchmarks for this particular problem. It is important to note this collection is not limited to machine learning purposes.

1.1 Core

1. The creation of a mixed-integer linear programming (MILP) solver. It will use LocalSolver and its Python API. The solver is used to solve the instances provided by the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS).
2. Performing feature identification and computation to extract the features from the raw data (DIMACS) and to achieve the ground truth for the instances.
3. Performing feature selection to select a subset of relevant and useful features for the construction of the model.
4. Using the selected features and ground truth to train a classification model to learn the most optimal solution to CVRP.
5. Evaluating the performance of the model based on the selected criterion.

1.2 Advanced

1. The improvement of the model performance through methodical feature identification, computation, and selection and parameter tuning.

Chapter 2: **Introduction**

Combinatorial optimization has been around for decades with many different methods used to solve this difficult domain of problems. For example, exact algorithms or heuristics. Exact algorithms provide the optimal solution but requires a long running time [1]. On the other hand, heuristics are fast and accurate but do not guarantee optimality and require specialist knowledge [1]. Therefore, achieving the optimal solution in a short amount of time is preferred.

The vehicle routing problem (VRP) is a combinatorial optimization and integer programming problem that generalises the famous travelling salesman problem. The VRP is concerned with finding a set of routes for a fleet of vehicles to traverse in order to deliver to a given set of customers [2–4]. In the area of the VRP, there are many variants. This project focuses on the capacitated vehicle routing problem (CVRP) variant where the vehicles have a limited capacity.

Recently, neural networks and deep learning are used to tackle the VRP variants [5]. In contrast to both the traditional and modern techniques, this project aims to leverage classification as it is efficient, scalable, and does not lose optimality. The goal is to prune edges of the CVRP that are not part of the optimal solution to decrease the running times while maintaining optimality for exact algorithms. As the problem is NP-hard, any reduction in the problem size will result in evident improvements in the run time given a large enough problem.

Classification models are trained in order to prune edges of the CVRP problem. The training and test dataset consists of edges of the CVRP benchmark instances from DIMACS. These edges contain features where the models learn to classify between an optimal edge and a non-optimal one. The models are evaluated for a high recall and a high pruning rate. Then the best performing model is selected to prune unseen instances. The pruning process attempts to decrease the running time for the mixed-integer linear programming solver while maintaining optimality.

In this project, I proposed the method to apply classification to prune the non-optimal edges of the CVRP instances to sparsify the graph in order to reduce the running time of the solver while maintaining optimality. Chapter 3 explores the related works and the contribution of our project to this domain. Chapter 4 explains the data considerations and how the data pre-processing. Chapter 5 is the outline of the approach which explains briefly about the three core sections of this project. The first core section is in chapter 6 which explains the mixed-integer linear programming (MILP) formulations. The second core section is in chapter 7 which explains the classification process in terms of feature engineering, model training and hyperparameter tuning, and model evaluation. The third core section is in chapter 8 which explains the pruning evaluation and the results of this project. It showcases the best performing model, the random forest, is able to generalise and prune at least half of the edges of unseen CVRP instances, decrease the running time, and maintain optimality. Furthermore, for certain instances, it reduced the running time from 300 seconds to less than 10 seconds after pruning while maintaining optimality. It demonstrates the proposed method in this project is efficient, scalable, does not lose optimality, and reduces the running time of exact algorithms. Finally, chapter 9 summarises the project.

Chapter 3: Related Work and Ideas

Firstly, it is a discussion of current approaches to using machine learning on data structures and algorithm design. After, it is a discussion of using machine learning for combinatorial optimization problems. Thirdly, it is a discussion of how combinatorial optimization problems are generally solved. Lastly, it is a discussion of using machine learning for CVRP.

3.1 ML for Data Structures and Algorithms Design

Using machine learning for data structure and algorithm design is otherwise known as learning augmented algorithms. The implementation of data structures to use machine learning to adapt to input data [7]. The following shows the two classes of learning algorithms and generally how every caliber of algorithms can leverage machine learning to adapt to the input data, overcome the assumption built into such data structures and algorithms, and exploit common data patterns. The correctness and optimality of data structures are still maintained [7]. After, the leverage of machine learning for online algorithms explores the advantages further.

3.1.1 CDF-based learned algorithms

Specifically, a CDF-based model learns the sorting order or lookup keys structure to predict the position or existence of entries [8]. However, using neural networks did not work as conventional model architectures are too expensive and machine learning frameworks are not designed to operate in nanoseconds [8, 9]. To overcome these problems, Kraska et al. [8] developed the learning index framework (LIF), recursive-model indexes (RMI), and standard-error-based search strategies. The results are optimistic as for example, in the case of the learned index for integer datasets for a B-tree, the learned index performs significantly better in majority settings by 1.5 to 3 times faster while being two orders-of-magnitude smaller [8]. A slight deterioration is witnessed with string datasets as they are generally more expensive to search over [8]. In general, most data structures based on the cumulative distribution function can be improved through learning and promises a profuse potential [8, 9].

3.1.2 Oracle / Full-model learned algorithms

Specifically, in the case of the oracle, it learns prediction models. In the case of the full-model, it learns the entire data structure or algorithm [9]. Even in the case of inaccurate machine learning models, it still improved the performance of the algorithms used to solve problems like the approximate set membership and job-scheduling problem as we will see below [10]. Mitzenmacher et al. [10] used machine learning to learn the Bloom filter, and thereby scaling faster than conventional Bloom filters, resulting in a better space complexity [9]. As mentioned at the beginning of this chapter, the reason this is possible is due to the core aspect that machine learning can identify structure in the data.

3.1.3 Online algorithms

Similar to the above, in online algorithms, the utilization of machine learning is to find patterns in past data to improve the predictability of future data. Online algorithms deal with uncertainty regarding future input data while providing a worst-case guarantee and so benefits from using machine learning. As long as the following properties were present when using online algorithms with machine learning:

1. **Independence:** the algorithm is independent of the predictor
2. **Consistency:** algorithm performance improves with better predictions
3. **Robustness:** algorithm performance is bounded

then, verifiably, the algorithms worst-case improved [11, 12]. A simple example presented by Kumar et al [11]. showed that with the usage of machine learning, even in the case where the predictor was of bad performance, the performance of the binary search algorithm was upper-bounded by O(log n).

3.1.4 Using ML for more difficult problems

We can see here that the basis of using machine learning was not to overtake traditional approaches but rather, serve as a supporting element to improve current state-of-the-art methods such as lowering the time or space complexity [7]. Given so, can we take the same thought process and apply it to more difficult computer science problems?

3.2 ML for Combinatorial Optimization

The motivations for using machine learning on combinatorial optimization problems are [5]:

1. to replace heavy computation with a fast approximation based on assuming specialized knowledge on the optimization algorithm.
2. oppositely, the assumption of such knowledge may not be sufficient, and so there is a need to search for the best performing policy.

Three categories represent the existing techniques for machine learning on combinatorial optimization problems.

3.2.1 Supervised learning for combinatorial optimization

A recurrent algorithm in the space of combinatorial optimization is Branch-and-Bound. The variable options to branch upon determines the efficiency of the optimization [5, 13]. Strong branching has good performance, but computationally expensive heuristic [5, 13]. Therefore, many similar approaches use supervised learning to approximate strong branching decisions [5]. For instance, Alvarez et al. [13] fed as input to the machine learning model features describing the instances and the state of the branch-and-bound process [5]. The approximated function is hence produced [13]. In particular, in cases where the model is not performant for certain arbitrary instances, then traditional branch-and-bound is used [5]. The measurements in figure 2.1 show that it speeds up such decision process, but the performance is still behind traditional methods [13]. In these tables, 'gap' indicates how close the optimization is to finish, where the gap is between 0 to 1, 1 representing completion. 'term' constitutes the number of problems that terminated within the given nodes instead of the time limit. 'Nodes' and 'time' respectively represent the number of explored nodes and the time spent in seconds before the optimization either finds the optimal solution or stops earlier because of one terminating criterion [13].

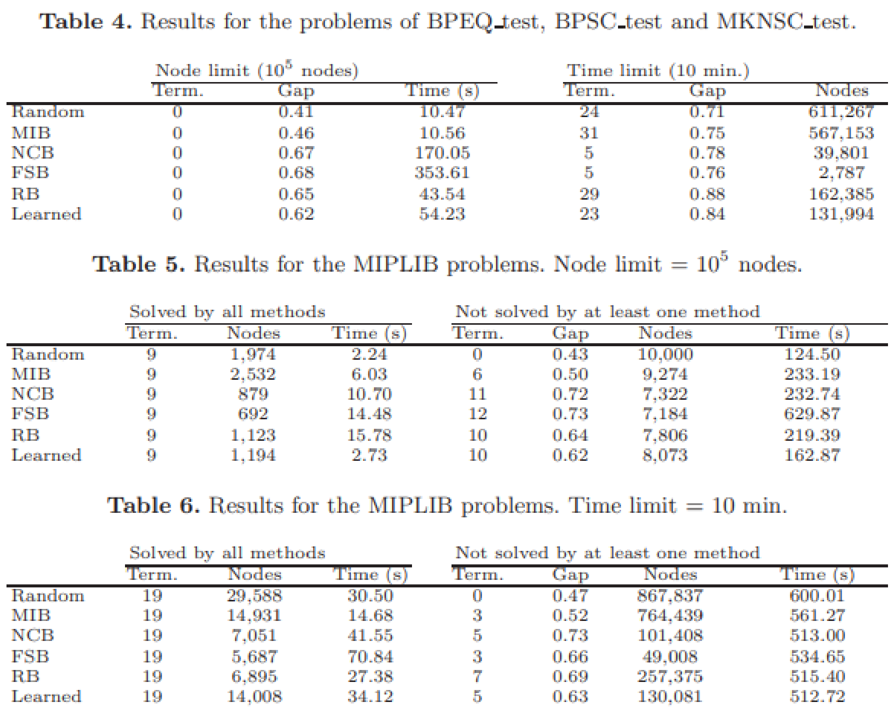


Table 4. Results for the problems of BPEQ_test, BPSC_test and MKNSC_test.

	Node limit (10^5 nodes)			Time limit (10 min.)		
	Term.	Gap	Time (s)	Term.	Gap	Nodes
Random	0	0.41	10.47	24	0.71	611,267
MIB	0	0.46	10.56	31	0.75	567,153
NCB	0	0.67	170.05	5	0.78	39,801
FSB	0	0.68	353.61	5	0.76	2,787
RB	0	0.65	43.54	29	0.88	162,385
Learned	0	0.62	54.23	23	0.84	131,994

Table 5. Results for the MIPLIB problems. Node limit = 10^5 nodes.

	Solved by all methods			Not solved by at least one method			
	Term.	Nodes	Time (s)	Term.	Gap	Nodes	Time (s)
Random	9	1,974	2.24	0	0.43	10,000	124.50
MIB	9	2,532	6.03	6	0.50	9,274	233.19
NCB	9	879	10.70	11	0.72	7,322	232.74
FSB	9	692	14.48	12	0.73	7,184	629.87
RB	9	1,123	15.78	10	0.64	7,806	219.39
Learned	9	1,194	2.73	10	0.62	8,073	162.87

Table 6. Results for the MIPLIB problems. Time limit = 10 min.

	Solved by all methods			Not solved by at least one method			
	Term.	Nodes	Time (s)	Term.	Gap	Nodes	Time (s)
Random	19	29,588	30.50	0	0.47	867,837	600.01
MIB	19	14,931	14.68	3	0.52	764,439	561.27
NCB	19	7,051	41.55	5	0.73	101,408	513.00
FSB	19	5,687	70.84	3	0.66	49,008	534.65
RB	19	6,895	27.38	7	0.69	257,375	515.40
Learned	19	14,008	34.12	5	0.63	130,081	512.72

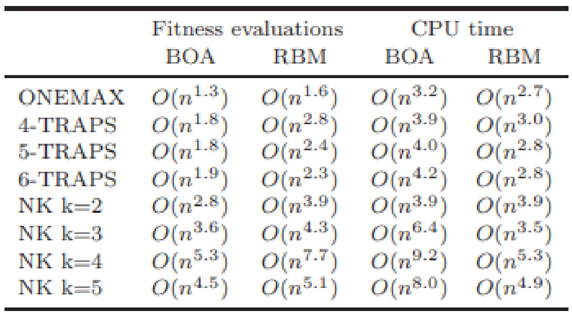
Figure 3.1: Table showing the results

Semidefinite programming (SDP) is another recurrent algorithm in the space of combinatorial optimization. Baltean-Lugojan et al. [14] used supervised learning to approximate the objective function of the SDP. The performance measures show that practically, sufficiently trained estimators are good models for the selection process. Like above, the model can select the best submatrices (optimal solution) without the heavy computation [5].

Essentially, the interpretation of such methods is learning the output from the input sequence where there is a usage of neural networks for example. However, these techniques require large training sets that need NP-hard problems to be solved, hence limiting the usefulness.

3.2.2 Unsupervised learning for combinatorial optimization

Combinatorial optimization problems see the usage of restricted Boltzmann machines. For example, the Estimation of Distribution Algorithm (RBM-EDA) by Probst et al. [15]. This method builds information about the probability distribution of satisfiable potential solutions. Results in figure 2.2 show that despite being less efficient in terms of fitness, RDM-EDA poses an attractive alternative to large, complex problems, and there is low eligibility evaluation computation is RDM-EDA [15].



	Fitness evaluations		CPU time	
	BOA	RBM	BOA	RBM
ONEMAX	$O(n^{1.3})$	$O(n^{1.6})$	$O(n^{3.2})$	$O(n^{2.7})$
4-TRAPS	$O(n^{1.8})$	$O(n^{2.8})$	$O(n^{3.9})$	$O(n^{3.0})$
5-TRAPS	$O(n^{1.8})$	$O(n^{2.4})$	$O(n^{4.0})$	$O(n^{2.8})$
6-TRAPS	$O(n^{1.9})$	$O(n^{2.3})$	$O(n^{4.2})$	$O(n^{2.8})$
NK k=2	$O(n^{2.8})$	$O(n^{3.9})$	$O(n^{3.9})$	$O(n^{3.9})$
NK k=3	$O(n^{3.6})$	$O(n^{4.3})$	$O(n^{6.4})$	$O(n^{3.5})$
NK k=4	$O(n^{5.3})$	$O(n^{7.7})$	$O(n^{9.2})$	$O(n^{5.3})$
NK k=5	$O(n^{4.5})$	$O(n^{5.1})$	$O(n^{8.0})$	$O(n^{4.9})$

Figure 3.2: Approximated scaling behavior between the number of fitness evaluations, problem size, CPU times, and problem size

3.2.3 Reinforcement learning for combinatorial optimization

The interpretation of combinatorial problems for reinforcement learning is learning useful heuristics [16]. The measurement of the caliber of the solution is through the computation of the optimization objective.

Neural Monte Carlo Tree Search with Self-play

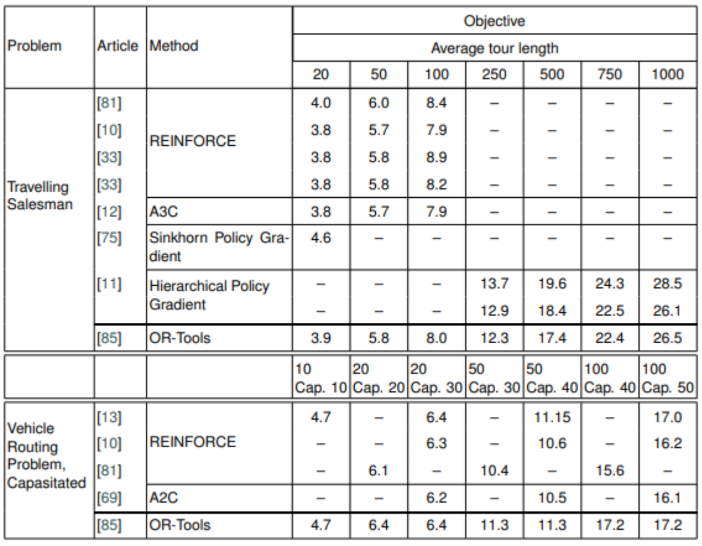
Laterre et al. [17] propose a model-based algorithm where it uses deep neural networks, tree search, and self-play where it performs significantly better than competing alternatives such as the Lego heuristic. Neural MCTS is used prominently in the use of reinforcement learning for combinatorial optimization problems. The different solutions use variants of the Neural MCTS by modifying the graph or modifying the algorithm itself [5, 16]. For example, in the 2D and 3D bin packing problem, Neural MCTS is combined with a learnable policy and value function [16].

Policy-based

REINFORCE algorithm by Williams [18] is a very performant algorithm in comparison to its peers [16]. A pointer network architecture encodes the input sequence, and the achievement of the sequential formulation of the solution is from a distribution over the input using the pointer process of the decoder. It is then in parallel, asynchronously trained. REINFORCE allows for the other more effective reinforcement algorithms to build on top of it. As it is a gradient-based method, it integrates well with other gradient computation techniques but is vulnerable to converge to a false optimum.

Value-based

Deep Q-Network (DQN) is a popular method to solve combinatorial optimization tasks where the RL algorithms, in general, use Q-learning to find the optimal solutions [16]. For example, in Dai et al. [19], the parameterization of the Q-function is via a deep graph embedding network. The achievement of backpropagation is through the loss function at each training step (reinforcement learning). The end-to-end framework automatically designs greedy heuristics for difficult combinatorial optimization problems on graphs. The results prove the consistent performance of the framework and its effectiveness in comparison to manually crafted algorithms.



Problem	Article	Method	Objective						
			Average tour length						
			20	50	100	250	500	750	1000
Travelling Salesman	[81]	REINFORCE	4.0	6.0	8.4	–	–	–	–
	[10]		3.8	5.7	7.9	–	–	–	–
	[33]		3.8	5.8	8.9	–	–	–	–
	[33]		3.8	5.8	8.2	–	–	–	–
	[12]	A3C	3.8	5.7	7.9	–	–	–	–
	[75]	Sinkhorn Policy Gradient	4.6	–	–	–	–	–	–
	[11]	Hierarchical Policy Gradient	–	–	–	13.7	19.6	24.3	28.5
			–	–	–	12.9	18.4	22.5	26.1
	[85]	OR-Tools	3.9	5.8	8.0	12.3	17.4	22.4	26.5
			10 Cap. 10	20 Cap. 20	20 Cap. 30	50 Cap. 30	50 Cap. 40	100 Cap. 40	100 Cap. 50
Vehicle Routing Problem, Capasitated	[13]	REINFORCE	4.7	–	6.4	–	11.15	–	17.0
	[10]		–	–	6.3	–	10.6	–	16.2
	[81]		–	6.1	–	10.4	–	15.6	–
	[69]	A2C	–	–	6.2	–	10.5	–	16.1
	[85]	OR-Tools	4.7	6.4	6.4	11.3	11.3	17.2	17.2

Figure 3.3: The comparison of average tour lengths for TSP and Capacitated VRP for Erdős–Rényi graphs

3.2.4 Promising approaches and their consequences

Despite the large complexity that goes into utilizing machine learning to solve more difficult problems, the results are promising more often than not [16]. However, it is important to note the prominent issues. For instance, attempting to learn the full solution not only sometimes yields poor results, but it is common that the neural networks are not interpretable, and a huge dataset with labeled instances are required [13]. Furthermore, in the cases of using reinforcement learning where it overcomes the said issues in the last point, it is often the case these approaches combine many different methods or modifies the graph itself [17]. Applying reinforcement learning increases the complexity of the model, still does not solve the issue of interpretability, and it is not clear whether a change in the dataset affects the learned model or not [6]. It brings about the question, what about using machine learning similar to the approaches for data structures and algorithm design, rather than trying to learn the whole solution?

3.3 Combinatorial Optimization

The usage of combinatorial techniques is present in combinatorial optimization. It allows for the determination of the best feasible solution from a finite set of possibilities. However, some problems are NP-hard, and so the difficulty resides in designing an exact efficient algorithm [20]. The interpretation of the VRP family is combinatorial due to the numerous solutions, which grows exponentially with the size of the instances, and also the variants grow exponentially with the problem domain's characteristics [21]. Current solutions suffer from long execution times and the need to rely on specialized knowledge to develop algorithmic rules for specific scenarios [1].

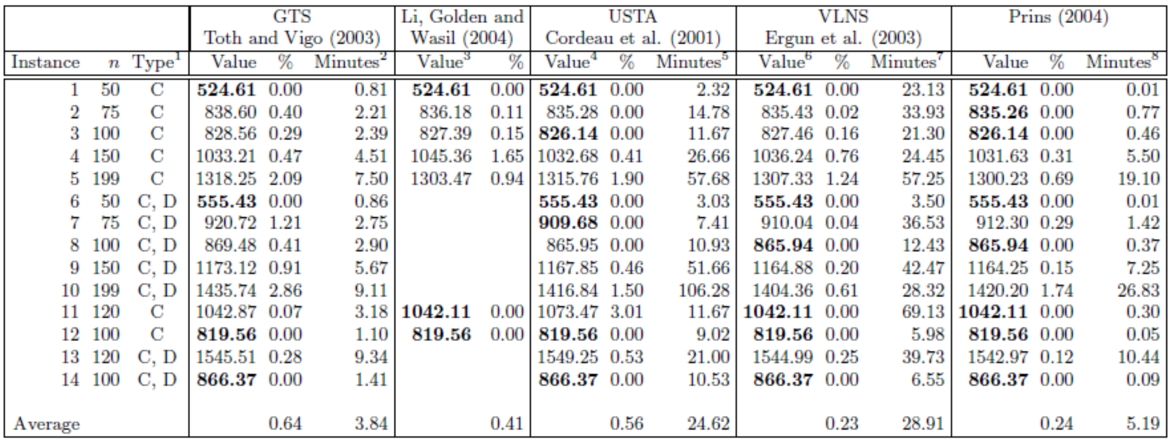
3.3.1 Heuristic algorithms for CVRP

The popularity and common usage of heuristics are because it is possible to use any heuristic in a large set of optimization problems [22]. Furthermore, it can usually find a solution relatively fast

[3]. The disadvantage is that the solution is not guaranteed to be optimal [3].

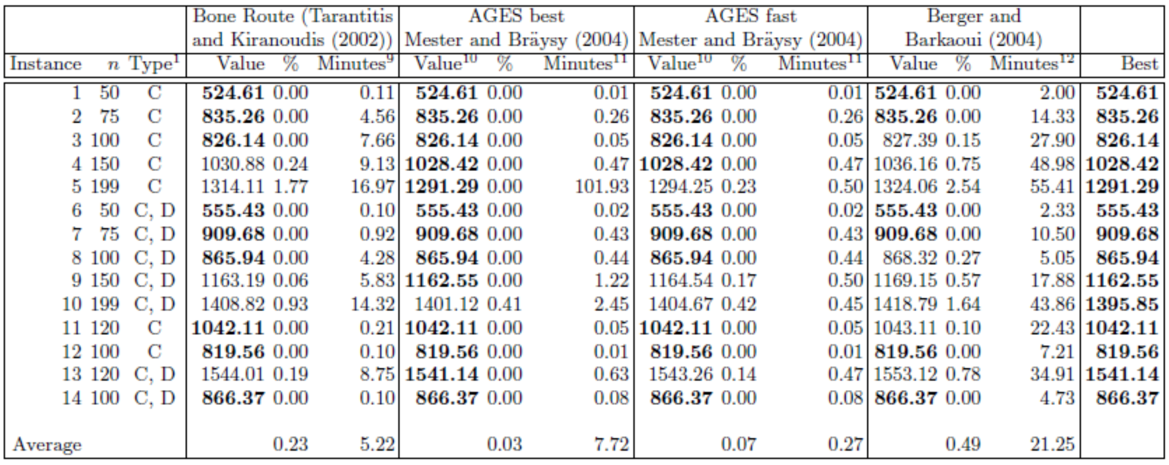
Metaheuristic

Compared to traditional heuristics, this is a more thorough search and less probable to end with a local optimum [23]. A significant research effort went into metaheuristics such as Genetic algorithms, Tabu search, Simulated annealing, and Adaptive Large Neighborhood Search (ALNS) due to the difficulty of finding the optimal solution for large instances of VRP. Lately, several efficient metaheuristics for VRP achieve solutions within 0.5% or 1% of the optimum for instances from hundreds to thousands of delivery points [24]. In essence, metaheuristics provide general frameworks for heuristics that apply to many problems [3]. They combine a variety of principles, and the best algorithms prove to be highly accurate, and some are fast [25]. The active guided evolution strategies (AGES) algorithm of Mester and Bräysy (2004) is currently the best metaheuristic [26]. Its deviation from the best-known value beats other algorithms by tenfold, see figure 2.5 [25]. In other words, its performance is the best in terms of accuracy and computation time. There is a large preference for the metaheuristic approach for large-scale applications with complex constraints.



			GTS Toth and Vigo (2003)			Li, Golden and Wasil (2004)		USTA Cordeau et al. (2001)			VLNS Ergun et al. (2003)			Prins (2004)		
Instance	n	Type[1]	Value	%	Minutes[2]	Value[3]	%	Value[4]	%	Minutes[5]	Value[6]	%	Minutes[7]	Value	%	Minutes[8]
1	50	C	**524.61**	0.00	0.81	**524.61**	0.00	**524.61**	0.00	2.32	**524.61**	0.00	23.13	**524.61**	0.00	0.01
2	75	C	838.60	0.40	2.21	836.18	0.11	835.28	0.00	14.78	835.43	0.02	33.93	**835.26**	0.00	0.77
3	100	C	828.56	0.29	2.39	827.39	0.15	**826.14**	0.00	11.67	827.46	0.16	21.30	**826.14**	0.00	0.46
4	150	C	1033.21	0.47	4.51	1045.36	1.65	1032.68	0.41	26.66	1036.24	0.76	24.45	1031.63	0.31	5.50
5	199	C	1318.25	2.09	7.50	1303.47	0.94	1315.76	1.90	57.68	1307.33	1.24	57.25	1300.23	0.69	19.10
6	50	C, D	**555.43**	0.00	0.86			**555.43**	0.00	3.03	**555.43**	0.00	3.50	**555.43**	0.00	0.01
7	75	C, D	920.72	1.21	2.75			**909.68**	0.00	7.41	910.04	0.04	36.53	912.30	0.29	1.42
8	100	C, D	869.48	0.41	2.90			865.95	0.00	10.93	**865.94**	0.00	12.43	**865.94**	0.00	0.37
9	150	C, D	1173.12	0.91	5.67			1167.85	0.46	51.66	1164.88	0.20	42.47	1164.25	0.15	7.25
10	199	C, D	1435.74	2.86	9.11			1416.84	1.50	106.28	1404.36	0.61	28.32	1420.20	1.74	26.83
11	120	C	1042.87	0.07	3.18	**1042.11**	0.00	1073.47	3.01	11.67	**1042.11**	0.00	69.13	**1042.11**	0.00	0.30
12	100	C	**819.56**	0.00	1.10	**819.56**	0.00	**819.56**	0.00	9.02	**819.56**	0.00	5.98	**819.56**	0.00	0.05
13	120	C, D	1545.51	0.28	9.34			1549.25	0.53	21.00	1544.99	0.25	39.73	1542.97	0.12	10.44
14	100	C, D	**866.37**	0.00	1.41			**866.37**	0.00	10.53	**866.37**	0.00	6.55	**866.37**	0.00	0.09
Average				0.64	3.84		0.41		0.56	24.62		0.23	28.91		0.24	5.19

Figure 3.4: Computational results for the Christofides et al. (1979) instances



			Bone Route (Tarantitis and Kiranoudis (2002))			AGES best Mester and Bräysy (2004)			AGES fast Mester and Bräysy (2004)			Berger and Barkaoui (2004)			
Instance	n	Type[1]	Value	%	Minutes[9]	Value[10]	%	Minutes[11]	Value[10]	%	Minutes[11]	Value	%	Minutes[12]	Best
1	50	C	**524.61**	0.00	0.11	**524.61**	0.00	0.01	**524.61**	0.00	0.01	**524.61**	0.00	2.00	**524.61**
2	75	C	**835.26**	0.00	4.56	**835.26**	0.00	0.26	**835.26**	0.00	0.26	**835.26**	0.00	14.33	**835.26**
3	100	C	**826.14**	0.00	7.66	**826.14**	0.00	0.05	**826.14**	0.00	0.05	827.39	0.15	27.90	**826.14**
4	150	C	1030.88	0.24	9.13	**1028.42**	0.00	0.47	**1028.42**	0.00	0.47	1036.16	0.75	48.98	**1028.42**
5	199	C	1314.11	1.77	16.97	**1291.29**	0.00	101.93	1294.25	0.23	0.50	1324.06	2.54	55.41	**1291.29**
6	50	C, D	**555.43**	0.00	0.10	**555.43**	0.00	0.02	**555.43**	0.00	0.02	**555.43**	0.00	2.33	**555.43**
7	75	C, D	**909.68**	0.00	0.92	**909.68**	0.00	0.43	**909.68**	0.00	0.43	**909.68**	0.00	10.50	**909.68**
8	100	C, D	**865.94**	0.00	4.28	**865.94**	0.00	0.44	**865.94**	0.00	0.44	868.32	0.27	5.05	**865.94**
9	150	C, D	1163.19	0.06	5.83	**1162.55**	0.00	1.22	1164.54	0.17	0.50	1169.15	0.57	17.88	**1162.55**
10	199	C, D	1408.82	0.93	14.32	1401.12	0.41	2.45	1404.67	0.42	0.45	1418.79	1.64	43.86	**1395.85**
11	120	C	**1042.11**	0.00	0.21	**1042.11**	0.00	0.05	**1042.11**	0.00	0.05	1043.11	0.10	22.43	**1042.11**
12	100	C	**819.56**	0.00	0.10	**819.56**	0.00	0.01	**819.56**	0.00	0.01	**819.56**	0.00	7.21	**819.56**
13	120	C, D	1544.01	0.19	8.75	**1541.14**	0.00	0.63	1543.26	0.14	0.47	1553.12	0.78	34.91	**1541.14**
14	100	C, D	**866.37**	0.00	0.10	**866.37**	0.00	0.08	**866.37**	0.00	0.08	**866.37**	0.00	4.73	**866.37**
Average				0.23	5.22		0.03	7.72		0.07	0.27		0.49	21.25	

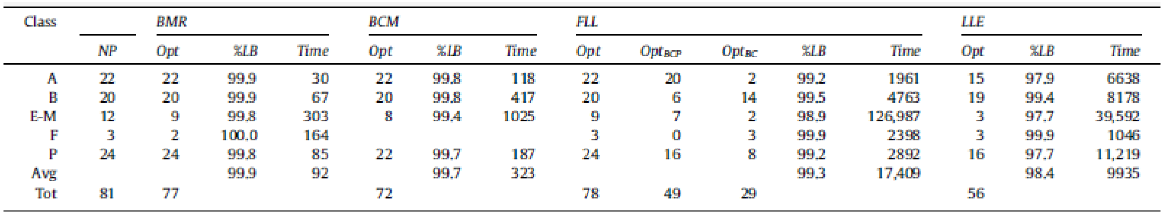
Figure 3.5: Computational results for the Christofides et al. (1979) instances

3.3.2 Approximation algorithms for CVRP

This algorithm is a class of heuristics that provide both a solution and an error guarantee [3]. Two types of approximation algorithms that can approximate the solution with any desired accuracy are polynomial-time approximation scheme (PTAS) and fully polynomial-time approximation scheme (FPTAS) [3]. However, for some problems, it is not possible to design a polynomial-time algorithm with constant error guarantee due to NP $\neq$ P, which questions whether PTAS and FPTAS are the best for NP-hard problems [3, 27].

3.3.3 Exact algorithms for CVRP

This method guarantees the optimal solution if it receives enough space and time. However, it is unexpected to construct this type of algorithm that solve NP-hard problems in polynomial time due to NP $\neq$ P [3]. However, as mentioned above, this is usually not the case. As of 2011, the most successful algorithms derive from the two-index flow formulation, the two-commodity flow formulation, and the set partitioning formula [4, 28, 29]. The general usage of the two-index formulation is for basic VRPs is limited in practicality as it only applies when the cost of the solution can be expressed as the sum of the costs of the arc costs [30]. In other words, this is not applicable when the cost or feasibility depends on a specific order of the customers or vehicles used. The two-commodity flow formulation uses additional integer variables with the edges to represent the flow of commodities [30]. The set partitioning formulation involves an exponential number of binary variables that represent different viable solutions and then deciding which solution has the minimum cost that satisfies the constraints [30]. In particular, the algorithm proposed by Baldacci et al. (in press) (BMR) on average performs faster than its competing algorithms [29], see figure 2.6.



Class		BMR			BCM			FLL					LLE		
	NP	*Opt*	*%LB*	*Time*	*Opt*	*%LB*	*Time*	*Opt*	*Opt*$_{BCP}$	*Opt*$_{BC}$	*%LB*	*Time*	*Opt*	*%LB*	*Time*
A	22	22	99.9	30	22	99.8	118	22	20	2	99.2	1961	15	97.9	6638
B	20	20	99.9	67	20	99.8	417	20	6	14	99.5	4763	19	99.4	8178
E-M	12	9	99.8	303	8	99.4	1025	9	7	2	98.9	126,987	3	97.7	39,592
F	3	2	100.0	164				3	0	3	99.9	2398	3	99.9	1046
P	24	24	99.8	85	22	99.7	187	24	16	8	99.2	2892	16	97.7	11,219
Avg			99.9	92		99.7	323				99.3	17,409		98.4	9935
Tot	81	77			72			78	49	29			56		

Figure 3.6: Computational comparison of the exact methods for the CVRP.

3.3.4 A place for ML

Highly researched traditional combinatorial optimization methods demonstrate that it is possible to produce an optimal solution with enough time or space. Or, solve the problem in the desired amount of time or space without optimality guarantee [3]. Consequently, is it possible to use machine learning to decrease the heavy computational work required of these algorithms to produce an optimal solution while maintaining interpretability and complexity?

3.4 ML for CVRP

Nazari et al. present a reinforcement learning framework that uses the Markov Decision Process (MDP) where optimal solutions are interpreted as decisions [31]. A parameterized stochastic policy and application of a policy gradient algorithm optimizes the parameters. Therefore the model produces the solutions as a sequence of real-time actions where it eliminates the need to re-train for each new problem instance. For medium-sized instances, results show that it outperforms traditional heuristics and Google's OR-Tools in terms of the quality of the solution and computation time. The approach allows for extension to other variants of the VRP and, more generally, combinatorial optimization problems. This approach is very familiar with the discussion in the above sections about the state-of-the-art frameworks.

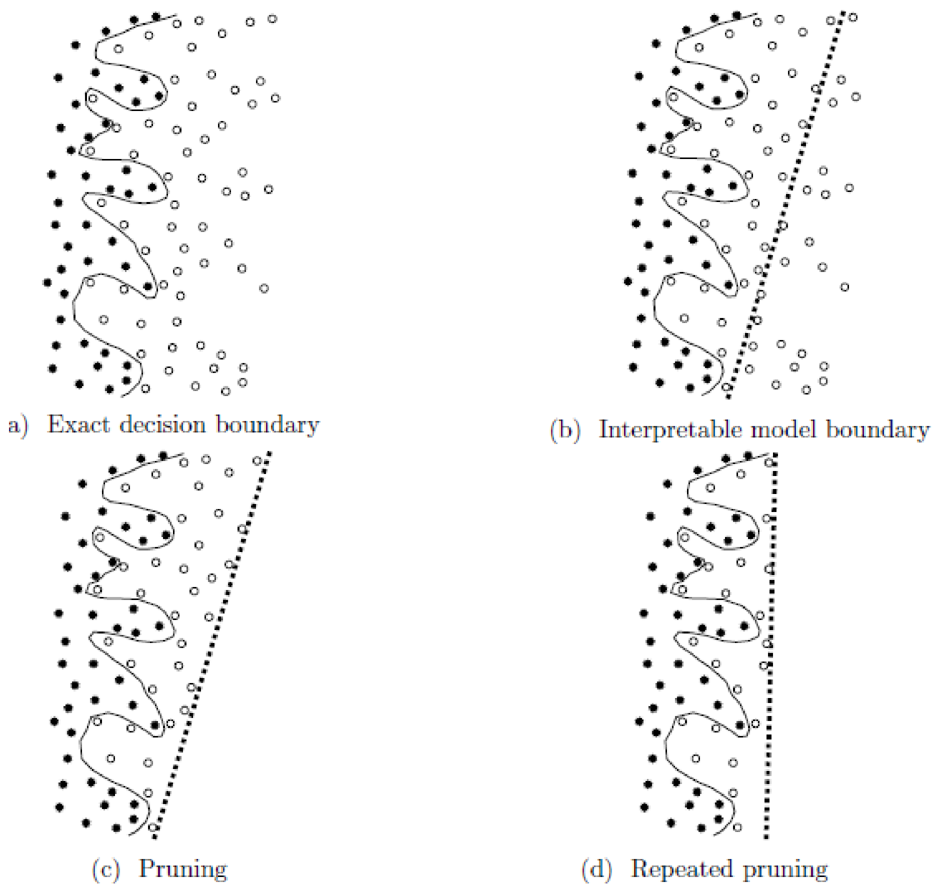


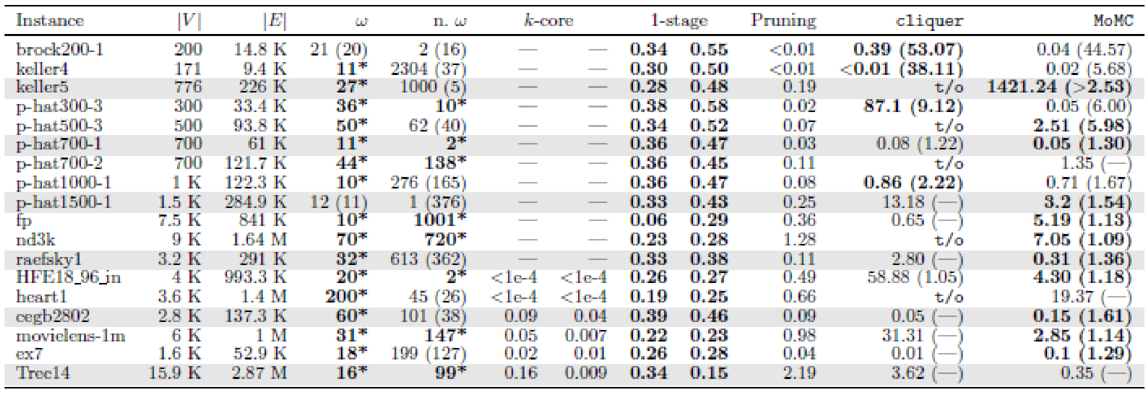
Figure 3.7: From Ajwani et al. Depiction of the overall framework. The black stars indicate the elements in the optimal subset while the white circles represent the elements not in the optimal subset. Earlier learning frameworks attempted to learn the exact decision boundary (drawn with a solid black line). The framework by Ajwani et al. uses a simple, interpretable classifier (shown by dashed lines) to repeatedly prune the white circles.

3.4.1 Limitations of the related works

Such approaches use a combination of different techniques, and as mentioned too, the successful methods apply a variety of modifications to either traditional algorithms or the data itself. Furthermore, they share a common sentiment where it is both complexes in terms of interpretability and architecture. It results in many different problems. The algorithm is hard to (mathematically) analyze, and the specific exploitation of the data is unclear [6]. Furthermore, in problems where (deep) supervised learning is used in an attempt to learn the entire solution, it demonstrates a challenging cycle where the lack of solved NP-hard problems result in fewer data to learn.

3.4.2 Using classification

Consequently, this project aims to approach NP-hard combinatorial optimization problems with a different perspective similar to Ajwani et al. [6] novel framework. The focus will be on exploiting local features for the machine learning models to learn from and hence produce a more intuitive and interpretable solution. Pruning of instances occurs if they are not in the optimal set. In the case of Ajwani et al. [6], the framework was able to prune a large portion of the graph yet maintaining most of the maximum cliques [6]. Furthermore, the results in show speedups in the solver figure 2.8. The framework is not only targeted at the maximum clique enumeration problem but designed to tackle the general problem of finding an optimal subset of instances. In any case, this allows better heuristics or algorithms to be designed and also lowers the heavy computational work of state-of-the-art solvers.



Instance	$\lvert V\rvert$	$\lvert E\rvert$	ω	n. ω	k-core		1-stage		Pruning	cliquer	MoMC
brock200-1	200	14.8 K	21 (20)	2 (16)	—	—	**0.34**	**0.55**	<0.01	**0.39 (53.07)**	0.04 (44.57)
keller4	171	9.4 K	**11***	2304 (37)	—	—	**0.30**	**0.50**	<0.01	**<0.01 (38.11)**	0.02 (5.68)
keller5	776	226 K	**27***	1000 (5)	—	—	**0.28**	**0.48**	0.19	t/o	**1421.24 (>2.53)**
p-hat300-3	300	33.4 K	**36***	**10***	—	—	**0.38**	**0.58**	0.02	**87.1 (9.12)**	0.05 (6.00)
p-hat500-3	500	93.8 K	**50***	62 (40)	—	—	**0.34**	**0.52**	0.07	t/o	**2.51 (5.98)**
p-hat700-1	700	61 K	**11***	**2***	—	—	**0.36**	**0.47**	0.03	0.08 (1.22)	**0.05 (1.30)**
p-hat700-2	700	121.7 K	**44***	**138***	—	—	**0.36**	**0.45**	0.11	t/o	1.35 (—)
p-hat1000-1	1 K	122.3 K	**10***	276 (165)	—	—	**0.36**	**0.47**	0.08	**0.86 (2.22)**	0.71 (1.67)
p-hat1500-1	1.5 K	284.9 K	12 (11)	1 (376)	—	—	**0.33**	**0.43**	0.25	13.18 (—)	**3.2 (1.54)**
fp	7.5 K	841 K	**10***	**1001***	—	—	**0.06**	**0.29**	0.36	0.65 (—)	**5.19 (1.13)**
nd3k	9 K	1.64 M	**70***	**720***	—	—	**0.23**	**0.28**	1.28	t/o	**7.05 (1.09)**
raefsky1	3.2 K	291 K	**32***	613 (362)	—	—	**0.33**	**0.38**	0.11	2.80 (—)	**0.31 (1.36)**
HFE18_96_in	4 K	993.3 K	**20***	**2***	<1e-4	<1e-4	**0.26**	**0.27**	0.49	58.88 (1.05)	**4.30 (1.18)**
heart1	3.6 K	1.4 M	**200***	45 (26)	<1e-4	<1e-4	**0.19**	**0.25**	0.66	t/o	19.37 (—)
cegb2802	2.8 K	137.3 K	**60***	101 (38)	0.09	0.04	**0.39**	**0.46**	0.09	0.05 (—)	**0.15 (1.61)**
movielens-1m	6 K	1 M	**31***	**147***	0.05	0.007	**0.22**	**0.23**	0.98	31.31 (—)	**2.85 (1.14)**
ex7	1.6 K	52.9 K	**18***	199 (127)	0.02	0.01	**0.26**	**0.28**	0.04	0.01 (—)	**0.1 (1.29)**
Trec14	15.9 K	2.87 M	**16***	**99***	0.16	0.009	**0.34**	**0.15**	2.19	3.62 (—)	0.35 (—)

Figure 3.8: The last two columns show the runtimes, where the number in the parenthesis indicate the speedup the solver achieved after the pruning of instances.

3.4.3 Considerations

Metaheuristics are not used as it does not provide an exact solution. It is necessary to solve for the exact solution in order

3.5 Summary

In conclusion, the benefits of using machine learning to aid in data structures and algorithm design was discussed. Additionally, how state-of-the-art research uses machine learning in combinatorial optimization problems was discussed. Following that, how combinatorial optimization problems are traditionally solved as well as state-of-the-art research approaches was discussed. Lastly, it was the discussion of the contribution of this project, and how it differs from current techniques.

Chapter 4: Data Considerations

4.1 Data

The engineered CVRP instances create the data to train and test the machine learning models. These instances are taken from http://dimacs.rutgers.edu/programs/challenge/vrp/cvrp/, a site that contains benchmark CVRP instances. These instances are currently used for a challenge competition.

4.1.1 Details of the CVRP Instances

Each instance file consists of:

- The euclidean 2D points that represent the customers and the single depot
- The respective identification number associated with each node (node ID)
- The demand associated with each node ID
- The number of vehicles
- The capacity number (the total demand available for the vehicle) of the vehicles
- The total number of nodes
- The instance name (e.g A-n32-k5) representing set A, 32 nodes, and 5 vehicles

These instances are the primary source of static data, where each instance is a structured text file and is stored locally after downloading. The instances are widely used, up-to-date, and currently used for a challenge competition. Furthermore, there are solutions to these instances which contain the optimal objective function value and the target labels. It was verified the solver returns the same target labels as the solution files.

4.1.2 Details of the Training, Test, and Validation Data

This project randomly selected 30 instances from a variety of CVRP sets. It consisted of eight instances from set A, eight instances from set B, seven instances from set E, and seven instances from set P. The data consists of edges computed from these CVRP instances. The scale of the data is huge as the edges of the graphs are of interest. Since it is a combinatorial problem, where the graph is a complete graph prior to calculating the solution, then there are $\frac{n(n-1)}{2}$ edges. Taking an instance file from set P with 14 nodes as an example, we have 91 edges. As the other instances have an increasingly larger number of nodes, the number of edges increases too, hence the large scale of data. The data used in this thesis was not gathered maliciously and is publicly available.

4.2 Pre-processing

The pre-processing of the data consisted of:

- Fixing naming inconsistencies in the CVRP instances.
- Generating training, test, and validation data for the machine learning models using the CVRP instances.

4.2.1 CVRP Instances

Some of the text files had inconsistent naming of the variables, and so it was required to create a consistent naming scheme to allow for the extraction of information.

4.2.2 Training, Test and Validation Data

For each CVRP instance file, all edges with its associated features were computed. The features marked with an asterisk (*) have been normalized by the number of nodes in that respective instance file. (see subsection 7.1.1). For an arbitrary pair of nodes (u, v) and E the set of edges, we computed its edges:

$$e1 = (u, v) \in E$$
$$e2 = (v, u) \in E$$

The positive examples of the training are all edges that are part of the optimal solution (routes of the CVRP), and the negative examples are edges that are not part of the solution. Due to the nature of the problem, there are more negative examples than positive examples, causing an imbalanced dataset. There were four ways to solve the imbalanced data problem:

1. Oversample the minority class to match the majority class.
2. Downsample the majority class to match the minority class.
3. Generate artificial positive examples using SMOTE.
4. Cost-sensitive learning.

Option one did not work as it introduced bias to the data. It tells the model there are more positive examples than there exists, and it performed poorly on the test and validation data that reflected the realistic scenario. Option three also did not work as the idea of an optimal and non-optimal edge may be similar. Option four was a viable option but, option two had the better performance. These experiments concluded that downsampling provided the best results. Furthermore, there were two ways to split the data:

1. The standard scikit-learn to perform the train test split.
2. Group the data by each instance, then perform the train test split.

Initially, the second method was used but produced inconsistent results depending on how the random seed shuffled and split by the instances. It is because each set has its characteristics. For example, set B nodes were grouped in clusters whereas, set A nodes are more randomly spread. The first method had a significantly better performance in both training, test, and validation. Therefore, the decision is to go with the first method. The splitting of the training data is 80% training and 20% testing during cross-validation. A stratified shuffle split was implemented when using the test data as it is imbalanced.

Finally, the dataset was rescaled using min-max normalisation as the features had different ranges.

The solution files provided the target labels.

Chapter 5: Outline of Approach

This project focuses on learning to prune the edges of the graph associated with the CVRP. The core concept is classification. The machine learning model learns to discriminate between edges. This approach is similar to the recently proposed framework by Ajwani et al. [6], where the framework prunes either the nodes or the edges. In that framework, it pruned instances of the maximum clique enumeration.

Our project builds upon the framework by tackling the CVRP. It focuses on the edges instead of the nodes as it is required to visit every node, therefore, pruning nodes is not possible.

5.1 Problem Statement

This paper focuses on is the Capacitated Vehicle Routing Problem (CVRP) where a fleet of kindred vehicles are available, and the concern is the vehicle's capacity [32, 33].

Formally, it is defined as [30, 32, 34, 35]:

$$Complete\ graph\ G = (V,\ E)\ where\ i \in V = \{0,\ 1,\ ...,\ V\}\ represents\ the\ ith\ vertex\ where:$$

$$V_i \begin{cases} customer & (i = 0) \\ depot & (i > 0) \end{cases}$$

$$and\ e_{i,j}\ \in\ E,\ i,\ j\ \in\ V\ and\ i \neq\ j\ represents\ the\ undirected\ edge\ from\ vertex\ i\ to\ j$$

A non-negative cost, $c_{i,j}$ is associated with each edge and represents the travel cost spent to go from vertex i to vertex j [2]. The goods demand to deliver to node i is q_i where, $q_i > 0$. The goal is to find Hamiltonian cycle(s) that satisfies:

1. $\forall\ q_i\ i \in V$
2. using all K vehicles
3. total demand of each vertex visited by a route does not surpass the vehicle capacity C

Essentially in this problem, the goal is to generate hamiltonian cycles from a given starting point where the number of cycles is equal to the number of vehicles specified while satisfying a range of constraints.

5.2 Mixed-Integer Linear Programming (MILP)

A MILP solver is programmed to solve the CVRP instances. The programmed MILP solver allows us to do the following:

1. Given it is a combinatorial optimization problem, validation is needed to determine if there are multiple unique optimal solutions as this improves the imbalanced dataset and determine what evaluation metric is viable. It was verified that there were no unique multiple optimal solutions, and so this project chose the recall evaluation metric.
2. Compute the linear relaxation and reduced cost features to use for the machine learning model training.
3. Run the solvers before and after pruning to measure the objective function value and the associated running time to evaluate this project.

5.3 Edge Framework

The edge framework compromises feature engineering the data for the model training and hyper-parameter tuning. After, the models are evaluated on a selection of metrics, and finally, the best performing model prunes the CVRP instance.

5.3.1 Feature Engineering

It involves generating features from the data consisting of edges and then performing feature importance and feature selection.

5.3.2 Model Training and Hyperparameter Tuning

The edge framework consists of the classifiers learning to prune the edges. It learns by looking at the edge characteristics defined by various features and prunes any edges of the graph that are not part of the optimal solution (negative labels).

5.3.3 Model Evaluation

Evaluation of the model involves:

1. ROC/AUC. It indicates how well the model distinguishes between an optimal and non-optimal edge.
2. The effect on the recall as we increase the prune percentage.

5.4 Pruning Evaluation

The best performing model prunes a selection of unseen CVRP instances. By adjusting the decision threshold, the best pruning rate that maintains optimality is found. Those parameters are used to prune a selected set of unseen instances. The measurements involve the objective function value and its associated running time before pruning, and then after pruning at the best pruning rate. It is expected to see the same objective function value and its associated running time to decrease.

5.5 Summary of the outlined approach

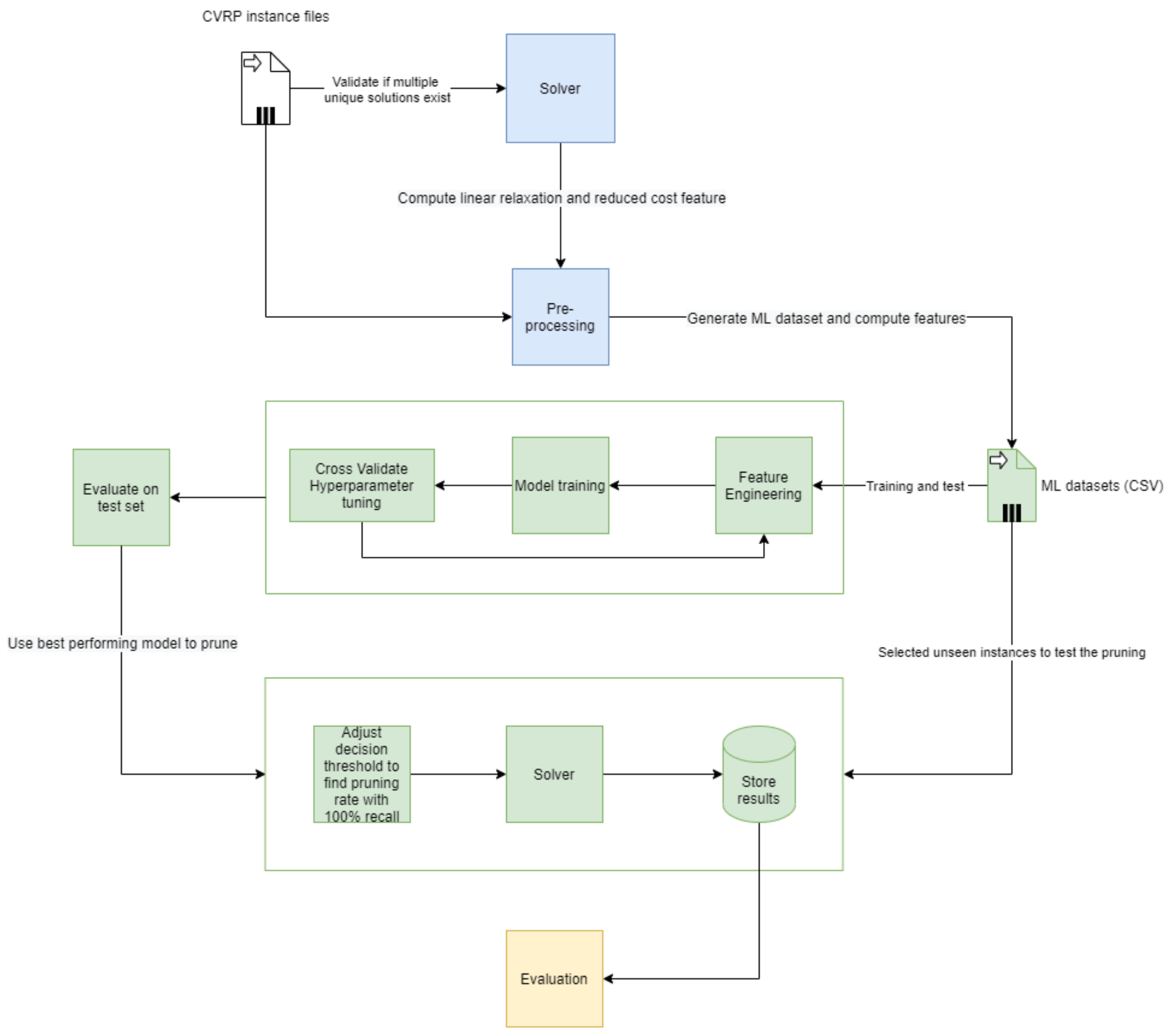

CVRP instance files
Validate if multiple unique solutions exist
Solver
Compute linear relaxation and reduced cost feature
Pre-processing
Generate ML dataset and compute features
ML datasets (CSV)
Training and test
Feature Engineering
Model training
Cross Validate Hyperparameter tuning
Evaluate on test set
Use best performing model to prune
Selected unseen instances to test the pruning
Adjust decision threshold to find pruning rate with 100% recall
Solver
Store results
Evaluation

Figure 5.1: Diagram showing architecture that summarises the outlined approach. Blue indicates the MILP process, green indicates the edge framework process, and yellow indicates the pruning evaluation process.

Chapter 6: Mixed-Integer Linear Programming (MILP)

Mixed-integer linear programming involves problems where some of the variables are integers while other variables are non-integers.

6.1 MILP Solvers

MILP solvers are optimization software. Optimization is the generation and selection of the best solution from some set of alternatives given a system transforming a set of inputs to outputs described by a function f. The solvers use Python as it is a convenient language, and has powerful modules like scikit-learn, numpy, and pandas that is helpful for machine learning. The Python APIs provided by the solvers are highly optimized [36–39] so the performance bottleneck only resides with the chosen formulations.

6.1.1 LocalSolver

LocalSolver is a global optimization solver, combining exact and heuristic techniques [36, 37]. This solver is used to solve the CVRP instances to measure the objective function value and its associated running time. This process occurs before and after pruning.

6.1.2 Gurobi

Gurobi, a mathematical programming solver [38] is used to calculate the linear relaxation and reduced cost feature. Gurobi uses a linear-programming-based branch-and-bound algorithm to solve mixed-integer linear programming problems [38].

6.2 Solver Solution Methods

As mentioned in the related works, there are three main approaches to solving the VRP, exact methods, approximation, and metaheuristics. This project follows an exact method approach as the goal is to achieve the optimal solution before and after pruning. It is important as it allows for an accurate evaluation of the pruning of the search space. The disadvantage is that it may be solved for small instances of the problem when compared to the other methods due to being an NP-hard problem [40].

6.2.1 MILP Solution Optimality

During the execution of the solver, it outputs the log every second indicating the current status of the solver, notably the optimality gap (difference between the best lower and upper bounds) and objective function value which is important.

Specifically, the optimality gap is the relative optimality gap. In other words, given r, it is a tolerance ranging from 0 to 1 indicating to the branch-and-bound solver that it should only search for solutions with objective values at least 100 * r% better than the current best solution[1]. It means if the gap is 5%, the solution is within 5% of the actual optimum. The typical gap is between 1% to 5%[2]. The smaller the optimality gap, the closer it is to the true optimum. The relative optimality gap can compute a solution within a few percentage points of the actual optimum after several minutes (assuming a relatively easy instance with an appropriate formulation) compared to several days.

In terms of the objective function value, the performance comparison is made with the value returned by the solver to the one provided in the instance files.

It is desirable to achieve a zero-gap and the same objective function value. However, this set of problems are notoriously difficult to solve. Therefore, there is a time limit of five minutes when running the pruning experiments. Chapter 8 contains further explanations and results.

In this paper, optimality refers to achieving the same objective function value as the one provided in the solution files and is within 0% to 5% of a gap. If the associated running time of the objective function value is less than 300 seconds (5 minutes), we achieved a 0% gap. Otherwise, it is between 0% to 5%.

6.2.2 Exact Solution Methods

There are three main approaches to solving the VRP:

1. Arc flow formulations (vehicle flow formulations) - Integer variables are associated with each arc counting the number of times that edge is traversed by a vehicle [41].
2. Commodity flow formulations - Additional - Extra integer variables are associated with each arc to represent the commodity flows of the traversed path [41].
3. Set partitioning problem - Have an exponential number of binary variables associated with a different feasible circuit where the VRP is a set partitioning problem that seeks for the collection of routes with the smallest cost while satisfying the VRP constraints [41].

Initially, this project followed the arc flow formulation approach using a two-index formulation. Literature mainly uses the two-index, or the three-index formulations [5]. A three-variable is not used as it has a lot of symmetry which makes it less viable for a branch-and-bound framework [5] (used by Gurobi). The performance in terms of running time and optimality deteriorated as the size of the problem increased. As this type of formulation is generally used for simple VRPs [41], and the data in this project are benchmark instances, the decrease in performance is expected. However, this solver implemented using Gurobi was still crucial as Gurobi allows for the computation of the linear relaxation (LR) feature and the reduced cost (RC) feature. This formulation also focuses on the individual edges which allows for the usage of the LR and RC features.

[1]Relative optimality gap: https://www.lindo.com/doc/online_help/lingo17_0/relative_optimality.htm
[2]Relative optimality gap: https://www.lindo.com/doc/online_help/lingo17_0/relative_optimality.htm

Subsequently, the set partitioning method was chosen to solve the instances as it performed better than the arc flow formulations. LocalSolver, the mathematical application provided much better documentation than Gurobi on implementing such formulations for the CVRP. However, unlike Gurobi, LocalSolver does not have the feature to compute the linear relaxation and reduced cost features [36, 37]. The formulation focuses on a subset of edges that form a set of routes rather than individual edges. Therefore, LocalSolver is used for solving the CVRP instances to measure the objective function value and its associated running time, and Gurobi calculates the linear relaxation and reduced cost features.

6.2.3 Performance Results

Tables 6.1 and 6.2 demonstrates both applications' performance on a variety of CVRP instances. Each column describes the following:

- **Application:** The solver application used.
- **Instance:** This is the name of the instance file. The file format is {set}-n{num_of_nodes}-k{num_of_vehicles}.
- **Running Time:** The running time of the solver. A maximum time limit of 300 seconds was set.
- **Solution:** Indicates optimal if the objective function value matches the one provided in the solution file, and the optimality gap is less than 5%. It is feasible otherwise.

Gurobi

Application	Instance	Running Time	Solution
Gurobi	A-n32-k5	300s	Feasible
Gurobi	B-n31-k5	300s	Feasible
Gurobi	E-n23-k3	6s	Feasible
Gurobi	P-n40-k5	300s	Feasible

Table 6.1: Performance of the solver using Gurobi.

LocalSolver

Application	Instance	Running Time	Solution
LocalSolver	A-n32-k5	9s	Optimal
LocalSolver	B-n31-k5	4s	Optimal
LocalSolver	E-n23-k3	1s	Optimal
LocalSolver	P-n40-k5	20s	Optimal

Table 6.2: Performance of the solver using LocalSolver.

Evidently, LocalSolver outperforms Gurobi. It is crucial as it allows for the accurate measurement of the objective function value and running time before and after pruning.

Summary

To summarise why both methods were adopted:

Application	Solver's Performance	Able to Calculate LRs and RCs	Relevant Documentation on CVRP
Gurobi	Feasible	**Yes**	No
LocalSolver	**Optimal**	No	**Yes**

6.3 MILP Formulation

The following subsections describe the mathematical formulation of the two index arc flow formulation, and the set partitioning formulation.

6.3.1 Two Index Arc Flow Formulation

The details of the variables are as follows:

- 0 is the depot
- n is the number of clients
- N is the set of clients, with $N = \{1, 2, ..., n\}$
- V is the set of nodes with $V = N \cup \ \{0\}$
- A is the set of arcs with $A = \{(i, j) \in V\}$
- c_{ij} is the travel cost over arc $(i, j) \in A$
- K is the number of vehicles
- Q is the vehicle capacity
- q_i is the amount required to deliver to the customer $i \in N$
- x_{ij} indicates if an arc is part of the optimal routes with $x_{ij} \in \{0, 1\} \quad \forall i, j \in V$

Then, we have the following formulation:

$$\min \sum_{i \in V, j \in V} c_{ij} x_{ij} \tag{6.1}$$

subject to:

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in N \tag{6.2}$$

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in N \tag{6.3}$$

$$\sum_{i \in V} x_{i0} = K \tag{6.4}$$

$$\sum_{j \in V} x_{0j} = K \tag{6.5}$$

$$u_j - u_i \geq q_j - Q(1 - x_{ij}) \qquad \forall i, j \in N, \;\; s.t. \;\; q_i + q_j \leq Q \tag{6.6}$$

$$q_i \leq u_i \leq Q \qquad \forall i \in N \tag{6.7}$$

Objective 5.1 minimizes the total travel cost [41]. Constraints 5.2 and 5.3 declare one and only one arc enters and leaves each node associated with a customer respectively [41]. Constraints 5.4 and 5.5 declares that there is an equal number of vehicles entering and leaving the depot respectively [41]. Constraints 5.6 is the route continuity and sub-tour elimination [41]. This ensures that the solution has no sub-tour disconnected from the depot. Furthermore, it declares that the vehicle capacity is a non-decreasing step function respective to the demand of the customers who are on the vehicle's route [41]. Constraint 5.7 ensures that u_i is restricted by an upper and lower capacity bound [41]. Setting the variable x_{ij} as a continuous type allowed us to compute the linear relaxation feature. The reduced cost feature is computed automatically and is retrieved via GRB.Attr.RC.

6.3.2 Set Partitioning Formulation

The details of the variables are as follows:

- R is the index set of routes with $R = \{1, 2, .., R\}$
- K is the number of vehicles
- a_{ir} indicates if a customer i is served by route r with $a_{ir} \in \{0, 1\}$
- x_r indicates if a route is selected with $x_r \in \{0, 1\}$ $\qquad \forall r \in R$

Then, the formulation follows:

$$\min \sum_{r \in R} c_r x_r \tag{6.8}$$

subject to:

$$\sum_{r \in R} a_{ir} x_r = 1 \qquad \forall i \in V \tag{6.9}$$

$$\sum_{r \in R} x_r \leq K \tag{6.10}$$

Objective 5.8 minimizes the total travel cost [41]. Constraint 5.9 states that a customer is served only by one route [41]. Constraint 5.10 states that the total number of routes is less or equal to the number of vehicles [41].

Chapter 7: **Edge Framework**

The edge framework learns to classify positive and negative edges by selecting engineered features where the evaluation of the performance decides the best model to prune CVRP instances. In this project, the classifiers chosen to use were the Random Forest, Logistic Regression, Gaussian Naive Bayes, and Support Vector Classifier. K-Nearest Neighbour was not chosen as it cannot perform feature importance. This project also tested Adaboosting and Gradient Boosting, but chosen not to continue with them as the Random Forest had significantly better performance. Out of these classifiers, the Random Forest had the best performance therefore the discussion in this chapter mainly focuses on the Random Forest.

7.1 Feature Engineering

The first part of the edge framework involves feature identification and computation and is then followed by feature selection.

7.1.1 Feature Identification and Computation

The computational of features is a core section of the project, and so extensive feature engineering went into this process. Computation of these features involved directly deriving from the provided dataset, following features discussed in literatures, general graph theory knowledge, and our intuition of the problem. The features marked with an asterisk (*) have been normalized by the number of nodes in that respective instance file. For an edge e, its respective nodes are denoted by u, v where the order is interchangeable as it is an undirected graph, meaning e = (u, v) = (v, u).

- **F1 - Is Node U Depot:** This feature is a binary variable directly taken from the instance file where the depot node has an ID of 1, and an integer 1 to indicate it is a depot. For the CVRP, there is only one depot. Of all the edges that contain the depot node, a subset of these edges are part of the optimal solution as they belong to either the beginning or end of the routes.
- **F2 - Is Node V Depot:** This is the same feature as F1 but for the other node of the edge.
- **F3 - Edge Weight*:** This feature is the Euclidean distance between two nodes, U, V that make up an edge. Given two nodes a, b, it is computed using numpy.linalg.norm(a - b). As the objective is to minimize the total travel cost, a feature that represents the cost to travel from node u to node v (and vice versa) is important. This feature also appeared frequently in literature [42–45].
- **F4 - Global Rank:** This feature is the ranking of the edge weights of all edges in each instance where a rank of 1 denotes the smallest weight. It is computed by sorting the weights from lowest to highest, with a tuple key representing the node IDs of the edges, and then transforming the weights into ranks. This feature indicates how valuable the edge is in terms of travel cost relative to the other edges.

- **F5 - Node U Local Edge Rank:** This feature is the ranking of the edge weights incident to each of the nodes. The computation follows the approach adopted in F6. Since it is a combinatorial optimization problem where the objective is to minimize the travel cost, this feature indicates the value of the travel cost for the next traversable edge.

- **F6 - Node V Local Edge Rank:** This is the same feature as F5 but for the other node of the edge.

- **F7 - Linear Relaxation*:** This feature is the transformation of our integer program into a related linear programming problem where the solution to this linear program is used to gain information about the solution to the original integer program [46]. Given x_{ij} indicating whether an edge is part of the solution, in the integer program, the constraint is of the form:

  $$x_{ij} \in \{0, 1\}$$

  whereas in the linear program, the constraint is of the form:

  $$0 \leq x_{ij} \leq 1$$

  This feature provides a tight lower bound on the solution quality of the original problem and provides information about the quality of edge in the original problem. It was computed using Gurobi by changing the variable x that indicates if an arc is part of the solution to a continuous attribute.

- **F8 - Node U Demand*:** This feature is the demand of the customer for each of the nodes. It is directly extracted from the instance file. In addition to minimizing the travel cost, the capacity of the vehicle must have enough space to satisfy the customers' demand. This feature was also used in Rasku et al. [45].

- **F9 - Node V Demand*:** This is the same feature as F8 but for the other node of the edge.

- **F10 - Node U Demand Capacity Ratio*:** This feature is the demand of the customer for each of the nodes divided by the constant capacity number of the vehicles. This feature indicates how much of the truck's capacity is used when visiting a customer at node u, or v. This feature was also used in Rasku et al. [45].

- **F11 - Node V Demand Capacity Ratio*:** This is the same feature as F10 but for the other node of the edge.

- **F12 - Node Average Demand Capacity Ratio:*** This is the average of F10 and F11. This feature is interpreted as how much of the truck's capacity is used when traversing this edge.

- **F13 - Reduced Cost:*** This feature is the amount by which the objective function coefficient would have to improve (decrease as an improvement in this project as the objective is to minimize) before the corresponding variable assumes a positive value in the optimal solution[1]. For this project, the reduced cost indicates how much of the travel cost between two nodes must be reduced before it is considered cost-effective (positive in the optimal solution). This feature was chosen based on the intuition that it is a good indicator of either a good or bad edge as the smaller the value, the more likely the edge is part of the optimal solution.

- **F14 - Node U Euclidean Distance to Nearest Convex Hull Vertex:*** This feature is the Euclidean distance for each of the nodes to the nearest vertex (node) that is part of the Convex hull. The computation of the distance follows the formula in F5. The convex hull is common in combinatorial optimization problems, path planning, and shape analysis [47] which is suitable to the problem. For this feature, the intuition is that the distance to these convex hull vertices provides a good indication of whether an edge is relevant. Since

[1]Reduced Cost: https://www.courses.psu.edu/for/for466w_mem14/Ch11/HTML/Sec4/ch11sec4_RC.htm

the convex hull is an enclosing area of the points, if the nearest convex hull vertex is further than the edge's current weight, then it indicates it is not a feasible traversable edge. This feature is also used in Rasku et al. [45].

- **F15 - Node V Euclidean Distance to Nearest Convex Hull Vertex:*** This is the same feature as F13 but for the other node of the edge.

- **F16 - Convex Hull Edge Weight:*** This feature is the Euclidean distance of an edge if the edge is part of the Convex hull. The computation of the distance follows the formula in F5. The Convex hull is calculated using Scipy's spatial ConvexHull module. Given a parameter of points, it generates a Convex hull. This feature was also used in Rasku et al. [45].

- **F17 - Node U Distance to Centroid:*** This feature is the Euclidean distance of each of the nodes to the centroid of the nodes. The computation of the distance follows the formula from F5. The centroid is calculated by summing all of the x coordinates and y coordinates, then dividing each by the number of coordinates to achieve a single (x, y) coordinate indicating the centroid. Rasku et al. [45] only calculate the distance to the centroid, so in this project, it was extended to calculating it for every node.

- **F18 - Node V Distance to Centroid:*** This is the same feature as F16 but for the other node of the edge.

- **F19 - Node Average Distance to Centroid:*** This is the average of F17 and F18.

- **F20 - Node U Distance to Depot:*** This feature is the Euclidean distance of each of the nodes to the depot. The computation of the distance follows the formula in F5. Nodes closest to the depot tend to start and end arbitrary routes of the solution. Furthermore, this feature has been shown to distinguish between optimal and sub-optimal solutions [43] and was used in literature [42–45].

- **F21 - Node V Distance to Depot:*** This is the same feature as F18 but for the other node of the edge.

- **F22 - Node Average Distance to Depot:*** This is the average of F20 and F21.

- **F23 - Node U Distance to Nearest Neighbour:*** This feature is the Euclidean distance of each of the nodes to its nearest neighbour. The computation of the distance follows the formula in F5. This feature was used in Rasku et al. and indicated it is a helpful feature [45].

- **F24 - Node V Distance to Nearest Neighbour:*** This is the same feature as F20 but for the other node of the edge.

- **F25 - Node Average Distance to Nearest Neighbour:*** This is the average of F23 and F24.

- **F26 - Angle Degree to Depot:** This feature is the angle of the depot vertex, where the vertices of the triangle consists of the two nodes of the current edge in question, and the depot. It is computed as follows:

```
if node_u == self.depot or node_v == self.depot:
    self.angle_degree_to_depot.append(0)
else:
    points = np.array([self.depot, node_u, node_v])
    A = points[1] - points[0]
    B = points[2] - points[1]
    C = points[0] - points[2]
```

```
angles = []
for e1, e2 in ((A, -B), (B, -C), (C, -A)):
    num = np.dot(e1, e2)
    denom = np.linalg.norm(e1) * np.linalg.norm(e2)
    if num == 0:
        angles.append(0)
    else:
        angles.append((np.arccos(round(num/denom, 6)) * 180) / np.pi)


self.angle_degree_to_depot.append(angles[0])
```

If the angle is wide, this indicates that the edge consisting of nodes u and v has a larger weight (node u and node v are far away from each other) which may suggest it is not an optimal edge as the objective is to minimize the travel cost. Furthermore, this feature has been shown to distinguish between optimal and sub-optimal solutions [43] and was used in literature [42–45].

- **F27 - Node U DBScan:*** This feature is a label of clusters of closely packed points. It is computed using sklearn's DBSCAN module. Nodes that are part of a cluster tend to belong to a particular route [45]. Intuitively, this feature is useful for instances where nodes are in clusters like set B.
- **F28 - Node V DBScan:*** This is the same feature as F23 but for the other node of the edge.
- **F29 - Node Average DBScan:*** This is the average of F27 and F28.
- **F30 - Is Optimal Edge:** This is the target label where 1 represents the edge as part of the solution.

It is important to note that the idea of an optimal and non-optimal edge may be similar, so features like F7 - Linear Relaxation and F13 - Reduced Cost (linear programming features) help differentiate such edges. Since different sets have different characteristics, features like F20 - Node U Distance to Depot (F21 and F22 too), F26 - Angle Degree to Depot, or any features that provide information about the graph by looking at the edges also help characterize optimal and non-optimal edges better [42, 44].

Features like the centrality, node degree, connectivity, minimum spanning tree were not particularly helpful. An improved performance was achieved after removing them. The intuition for this improvement is that the problem is a complete graph so features like the degree, centrality and connectivity do not provide information about optimal edges. An optimal edge has a degree of 2, whereas an edge in a complete graph has more. Centrality and connectivity intuitively do not make sense as it is required to visit every node once. Finally, minimum spanning tree only generates one subgraph without cycles, which is more suitable for the TSP.

7.1.2 Feature Importance and Feature Selection

Initially, the random forest selected the features to use for all models. However, models except random forest performed poorly. Allowing each model to individually select features achieved better results. Despite the performance improvement given the independent feature selection, the random forest still performed the best. Principal component analysis (PCA) was also tested but witnessed poorer performance when compared to without PCA. It is due to non-linear data. As mentioned in the outline of the approach, the feature selection involved using recall as the scoring metric.

Random Forest Feature Importance

The feature importance is calculated by fitting the training data to the random forest where it constructs many decision trees. The higher the value, the more important the feature. Below is a figure showing the feature importance computed by the Random Forest.

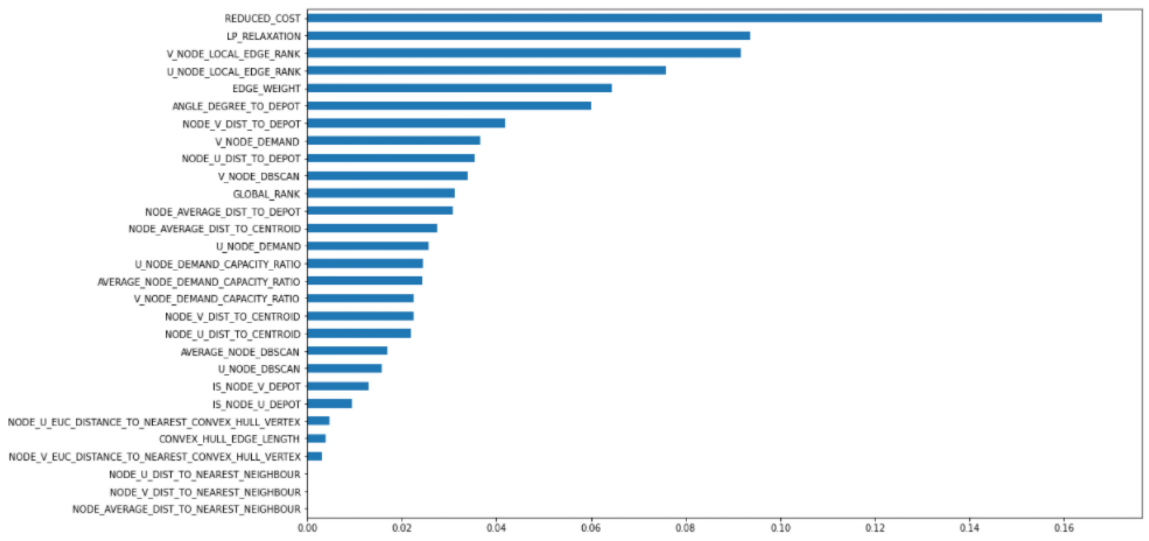


Figure 7.1: Random Forest Feature Importance.

Below are six features with relatively high feature importance 7.1.

- F13 - Reduced Cost
- F7 - Linear Relaxation
- F6 - Node V Local Edge Rank
- F5 - Node U Local Edge Rank
- F3 - Edge Weight
- F26 - Angle Degree to Depot

This feature importance ranking provides the following insights (concerning the random forest):

1. Feature F13 - Reduced Cost proves the intuition that it is a good indicator of whether an edge is part of the optimal solution.
2. Feature F26 - Angle Degree to Depot is a strong feature due to the ability to derive more information about the edge relative to the graph.
3. Concerning combinatorial optimization features, F6 - Node V Local Edge Rank and F5 - Node U Local Edge Rank are more important than F4 - Global Rank. Intuitively it makes sense as the objective is to find the best routes in a complete graph.
4. The convex hull and nearest neighbor features are not particularly helpful.

Random Forest Feature Selection

After ranking the features, a feature ranking with recursive feature elimination and cross-validated selection (sklearn RFECV) was performed to find the optimal number of features.

```
Optimal number of features : 23
```

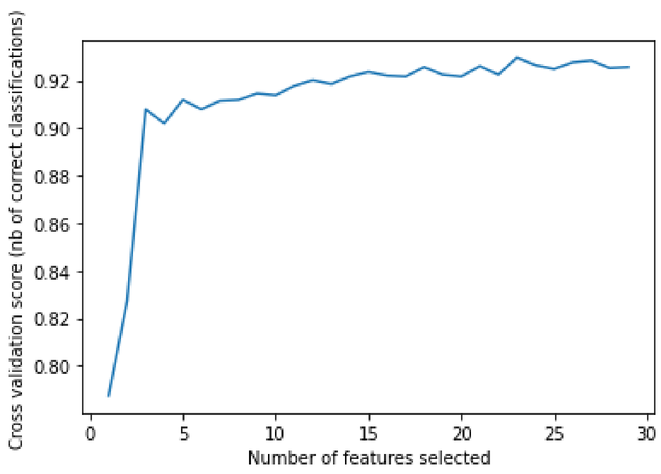
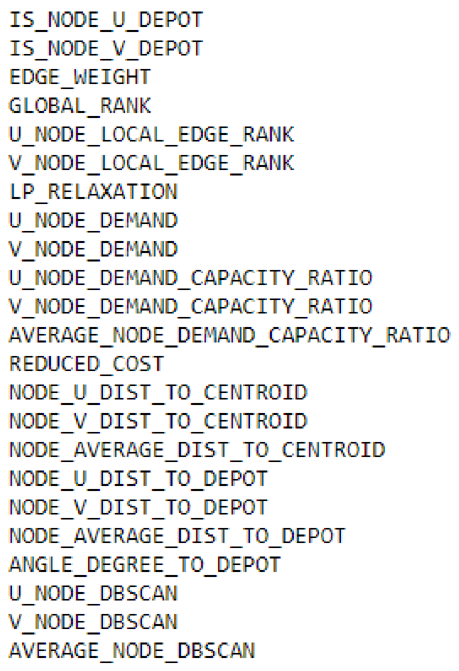


Figure 7.2: Random Forest recursive feature elimination and cross-validated selection.

The optimal number of features is 23, so feature ranking with recursive feature elimination (sklearn RFE) is utilized to find 23 features for the best model performance. The below figure 7.3 shows the selected features.



```
IS_NODE_U_DEPOT
IS_NODE_V_DEPOT
EDGE_WEIGHT
GLOBAL_RANK
U_NODE_LOCAL_EDGE_RANK
V_NODE_LOCAL_EDGE_RANK
LP_RELAXATION
U_NODE_DEMAND
V_NODE_DEMAND
U_NODE_DEMAND_CAPACITY_RATIO
V_NODE_DEMAND_CAPACITY_RATIO
AVERAGE_NODE_DEMAND_CAPACITY_RATIO
REDUCED_COST
NODE_U_DIST_TO_CENTROID
NODE_V_DIST_TO_CENTROID
NODE_AVERAGE_DIST_TO_CENTROID
NODE_U_DIST_TO_DEPOT
NODE_V_DIST_TO_DEPOT
NODE_AVERAGE_DIST_TO_DEPOT
ANGLE_DEGREE_TO_DEPOT
U_NODE_DBSCAN
V_NODE_DBSCAN
AVERAGE_NODE_DBSCAN
```

Figure 7.3: Random Forest Selected Features.

7.2 Model Training and Hyperparameter Tuning

After selecting the respective features, the models were trained and hyperparameter tuning was employed to improve the performance. A grid search with cross-validation was employed for all the models except for Gaussian Naive Bayes as it had no hyperparameters to tune. As mentioned, the discussion focuses only on the random forest.

Random Forest Hyperparameter Tuning

With hyperparameter tuning, it is possible to improve the pruning and recall of the best performing model when new instances were introduced for the pruning evaluation. Without hyperparameter tuning, it cannot prune as many edges as the recall decreases at a lower decision threshold. The improvement of the prune percentage was an average of 20% while maintaining 100% recall.

7.3 Model Evaluation

Finally, the models are evaluated to select the best performing model. The desired result is to have a ROC/AUC as close as possible to 1, and a consistently high recall as the prune percentage increases. Maintaining a high recall is important as it is important to minimize the number of positive edges pruned away. It is acceptable for the model to predict a negative edge as positive as after pruning, the solver runs on all positively predicted edges. During that process, it is able to find the true positive edges. The effect of predicting a non-optimal edge as optimal is a decrease in the improvement of the running time when compared to correctly predicting a non-optimal edge as non-optimal.

Prior to doing feature selection and hyperparameter tuning, a baseline comparison between all the models was carried out using all of the features and compared the recall scores. Despite these results, all four models were tuned, but the random forest model was expected to perform the best. As mentioned, the evaluation discussion focuses on the random forest model.

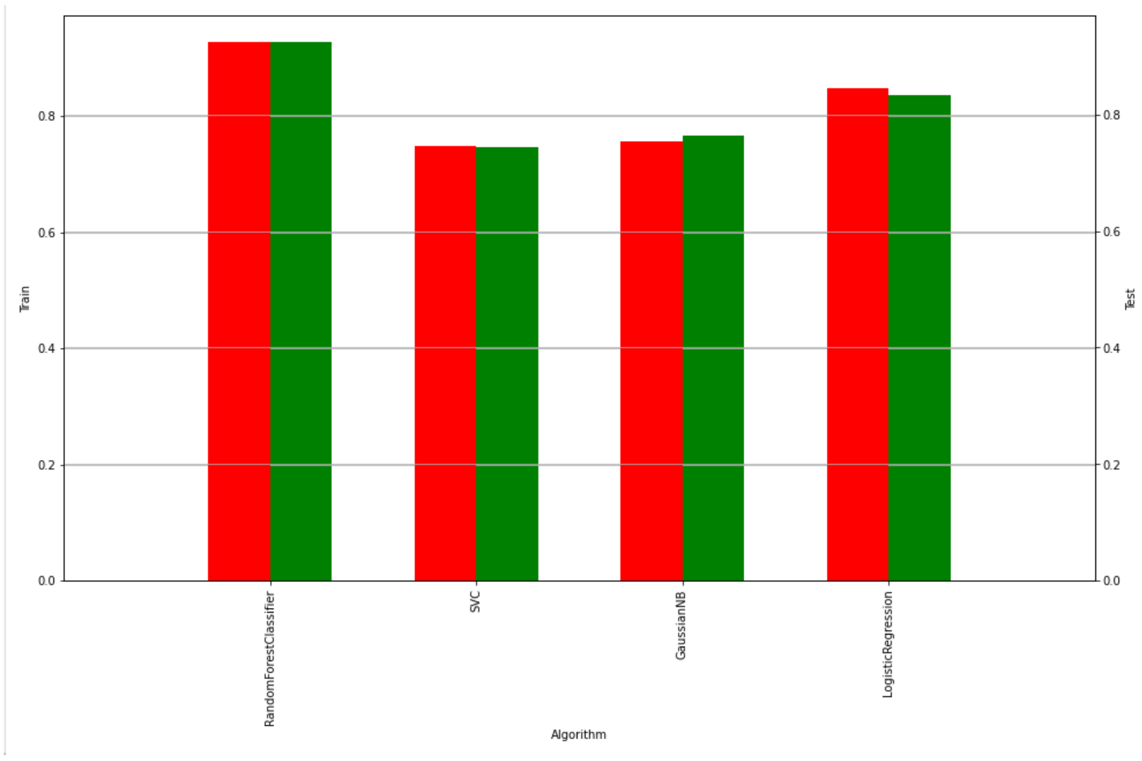


Figure 7.4: Comparing base models' recall performance using all features. Green bars represent the train scores, and the red represent the test scores.

Random Forest ROC/AUC

With the selected features and best parameters, we see here that the model is quite good at separating the two classes on the test data.

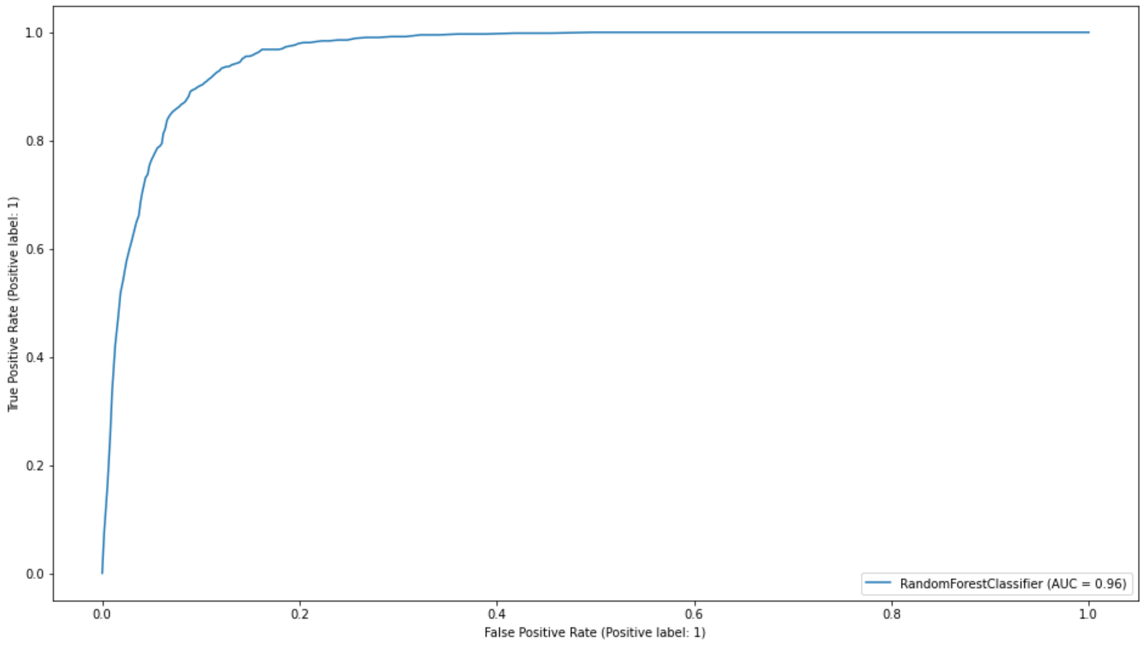


Figure 7.5: Random Forest ROC

Random Forest Recall vs Prune Percentage

Investigating deeper, it is possible to prune more than 50% of the edges and still maintain 100% recall. As mentioned, as the problem is NP-hard, any reduction in the problem size resulted in evident improvements in the run time given a large enough problem.

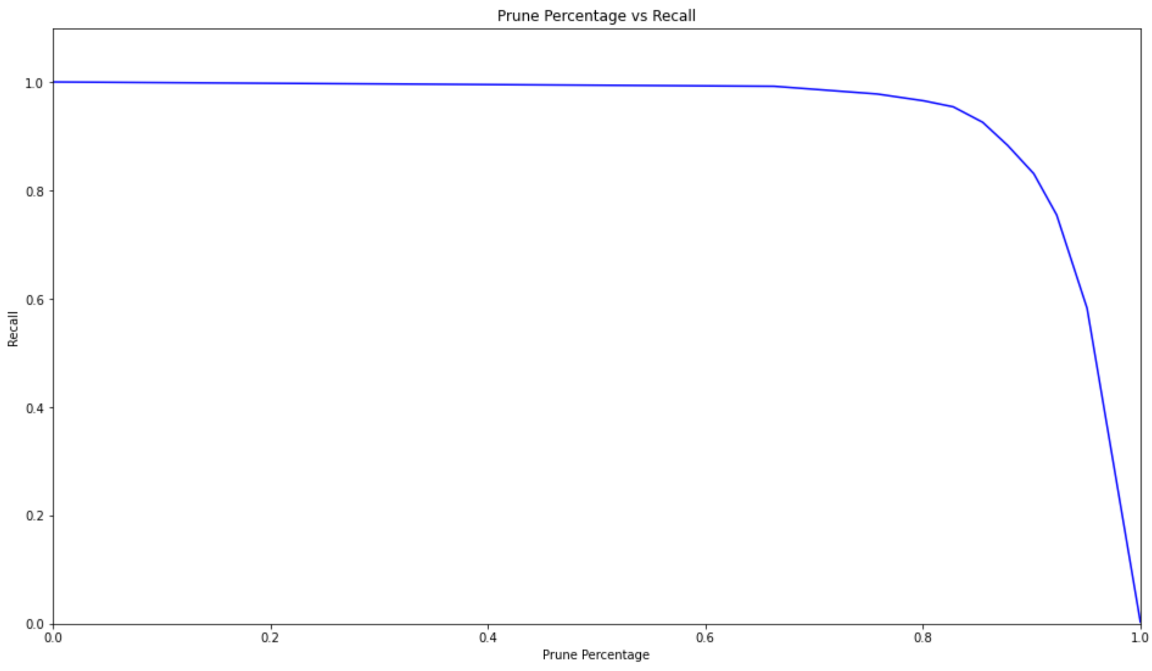


Figure 7.6: Random Forest Prune Percentage vs Recall.

Chapter 8: Pruning Evaluation

This chapter shows the performance figures of a selection of an unseen set of pruned instances. Overall, with the best possible pruning (a pruning rate where 100% recall is retained), it is possible to improve the running time while achieving the optimal solution. As the problem is NP-hard, any reduction in the problem size resulted in evident improvements in the run time given a large enough problem. The results showcased the model is able to generalise and prune at least half of the edges of unseen CVRP instances, decrease the running time, and maintain optimality. Furthermore, for certain instances, it reduced the running time from 300 seconds to less than 10 seconds after pruning while maintaining optimality. It demonstrates the proposed method in this project is efficient, scalable, does not lose optimality, and reduces the running time of exact algorithms.

8.1 Evaluation Metrics and Results

For each instance, the solver ran on the unmodified instance to measure the running time and objective function value. Then, the decision threshold of the Random Forest was varied to achieve 100% recall with its associated pruning rate. After, the model predicted the edges of the instance in question. Subsequently, the solver ran on the pruned instance where it consisted of positively predicted edges. With each iteration of this pruning process, the prune percentage, running time, and objective function value were recorded. This table 8.1 shows how many unseen instances was used per set. Set E has the least amount of instances, therefore, limiting the instances available to test.

Set	Instance Count
A	5
B	5
E	3
P	4

Table 8.1: Table to test captions and labels

8.1.1 Evaluation Metrics

Each column of the results tables describes the following:

- **Original Runtime:** The solver's duration in seconds to solve the unpruned instance. A maximum runtime of 300 seconds was set.
- **Pruned Runtime:** The solver's duration in seconds to solve the pruned instance. A maximum runtime of 300 seconds was set.
- **Original Gap:** The optimality gap for the unpruned instance. The optimality gap was set to have 0% tolerance.

- **Pruned Gap:** The optimality gap for the pruned instance. The optimality gap was set to have 0% tolerance.
- **Improvement:** The percentage decrease from the original gap to the pruned gap. It is calculated using:

$$Percentage\ of\ Improvement = \frac{(Original\ Gap\ -\ Pruned\ Gap)}{Original\ Gap} * 100$$

- **Prune Rate:** The percentage of edges pruned by the classifier.
- **ROC:** The receiver operating characteristic.

Each of the sets consisted of medium to large size instances and were difficult to solve (reached time limit). The average performance for each of the sets was computed. The below tables are our results.

8.2 Results

This project treated small instances to have less than 30 nodes (435 edges), medium instances to have 30 to 50 nodes (435 - 1,225 edges), and large instances to have anything above 50 (1,225 edges). The results demonstrates the proposed method in this project is efficient, scalable, does not lose optimality, and reduces the running time of exact algorithms.

Set A

Original Runtime	Pruned Runtime	Original Gap	Pruned Gap	Improvement	Prune Rate	ROC
300s	300s	3.32%	2.87%	16.13%	70%	0.96

Table 8.2: Set A Average Pruning Results.

This set consisted of randomly generated nodes within a square of side 100 and was a particularly difficult set to solve to 0% optimality gap as it took a long time. It is due to the constant vehicle capacity of 100 regardless of the number of nodes. With pruning, it converged more quickly to the 0% optimality gap, so it is expected that given enough time, the pruned instance will reach the 0% gap faster than the unpruned instance.

The experiments also tested smaller, easier-to-solve instances and witnessed a massive improvement. Before pruning, the solver took several seconds to minutes to solve the original instance and became instant after pruning (log returned 0 seconds) while achieving optimality.

These smaller tests were not included in our experiments. It is to highlight that it became more difficult to differentiate between an optimal and non-optimal edge as the instance became more complex. The complexity is because as the number of nodes increases, the graph became more densely packed with nodes due to the constant graph area. Therefore, in this common scenario, the insight is the decision threshold decreases which also causes a decrease in the improvement.

It was chosen not to include these tests to highlight that the idea of an optimal and non-optimal edge became very similar as the problem became more difficult. This difficulty is a result of the nodes becoming more dense. Therefore, the insight gained is the decision function threshold decreases as the number of nodes increases in such a situation which leads to a decrease in improvement.

Overall, the performance is very satisfactory as it is possible to achieve an improvement in performance across all instances, despite the similarity between an optimal and non-optimal edge in more difficult instances.

Set B

Original Runtime	Pruned Runtime	Original Gap	Pruned Gap	Improvement	Prune Rate	ROC
240.4s	181.6s	0.86%	0.67%	31.4%	50%	0.94

Table 8.3: Set B Average Pruning Results.

For small to medium-sized instances, set B is a lot easier to solve due to the nodes grouped in their respective clusters which inherently relates the clusters to a route. With pruning, it improved the running time significantly and reached closer to the 0% optimality gap. For this set, it achieved good performance for medium-size instances. It was possible to reduce the running time from 300 seconds to several seconds with a 0% optimality gap, while more difficult instances performed similarly to set A and E. Similarly to set A, smaller instances were also tested and witnessed similar results but did not include it to demonstrate the insight stated above.

Set E

Original Runtime	Pruned Runtime	Original Gap	Pruned Gap	Improvement	Prune Rate	ROC
300s	300s	5%	4.87%	3.05%	82%	0.98

Table 8.4: Set E Average Pruning Results.

Set E consisted of randomly generated nodes with a side of 70 and uniformly distributed nodes. This set was more difficult to solve than set A as it is more compact and is uniformly distributed rather than randomly distributed, hence the results. However, the situation is similar to set A where easier instances had a better performance, and is still able to improve the performance for difficult instances.

Set P

Original Runtime	Pruned Runtime	Original Gap	Pruned Gap	Improvement	Prune Rate	ROC
300s	226.5s	2.43%	1.86%	50%	75%	0.96

Table 8.5: Set P Average Pruning Results.

Set P consisted of a variety of modified instances based on the other sets. It had different performances that reflected the performances in sets A, B, and E. For example, it was possible to reduce the gap from 0.84% to 0.17%, or the runtime from 300 seconds to 6 seconds like set B. Other instances maintained at the 300 seconds runtime but saw a reduced optimality gap after pruning like set A, B, E. Overall, it is improved the performance greatly regardless of the instance.

Chapter 9: Summary and Future Work

9.1 Summary

CVRP is traditionally solved by using exact algorithms or heuristics [1]. Exact algorithms require long running times but provide optimal solutions [1]. Heuristics are quick, but do not guarantee optimality and requires specialist knowledge [1]. The goal of this project was to prune edges of the CVRP that are not part of the optimal solution to decrease the running times while maintaining optimality for exact algorithms. This project aimed to leverage classification as it is efficient, scalable, and does not lose optimality. As the problem is NP-hard, any reduction in the problem size resulted in evident improvements in the run time given a large enough problem. The classification models learned through features that captured the essence of whether an edge is part of the optimal solution. The best performing model, the random forest, then classified the edges of unseen CVRP instances to prune away edges that are not part of the solution. The results showcased the model is able to generalise and prune at least half of the edges of unseen CVRP instances, decrease the running time, and maintain optimality. Furthermore, for certain instances, it reduced the running time from 300 seconds to less than 10 seconds after pruning while maintaining optimality. It demonstrates the proposed method in this project is efficient, scalable, does not lose optimality, and reduces the running time of exact algorithms.

9.2 Future Work

- Improve the model by identifying features that characterise optimal and non-optimal edges better in hard instances.
- Train one model per instance set.
- Implement a stronger MILP formulation using Gurobi in order to move to one application. Alternatively, compute the linear relaxation and reduced cost features using LocalSolver if LocalSolver releases the features to do so.
- Test the pruning on the TSP since the VRP is a generalisation of the TSP.
- Implement neural networks to also prune instances of the CVRP as the methods follow very similarly.

Acknowledgements

I would like to deeply thank and appreciate the following people:

- Dr. Deepak Ajwani for his expert knowledge and amazing help.
- James Fitzpatrick for his advice on this project.
- Alan Fahey, my SAP manager, for his support, especially during difficult times.
- Robin Lee for letting me use his compute server.
- My friends and family for their support.

Appendix A: **Gantt Chart**

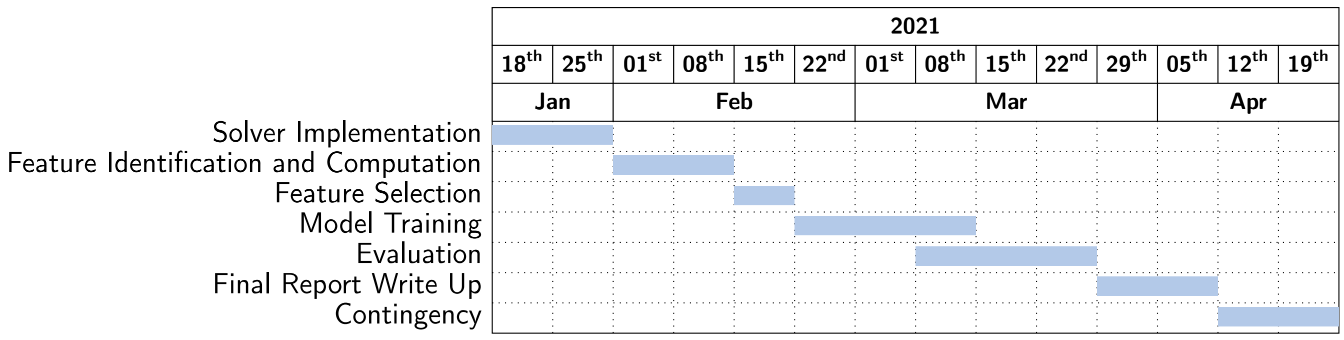


Figure A.1: Gantt Chart

The Gantt chart in figure A.1 presents the timeline of the project plan which includes both the completed work, and the remaining work of the project.

A.1 Detailed Descriptions

1. **Solver Implementation (2 weeks):**
   It involves coding a mixed linear programming formulation of CVRP using LocalSolver and its Python API. We will follow a set partitioning problem method where the objective function is given by:

   $$\min \sum_{r \in R} c_r x_r$$

   This finds a set of routes that minimizes the travel cost and number of vehicles used while satisfying various constraints associated with the problem.

2. **Feature Identification and Computation (2 weeks):**
   The DIMACS dataset includes the most optimal routes to the instances. Therefore, for each possible edge, the edge that is in the most optimal route should be labeled positive or negative otherwise. Furthermore, for each edge, a set of features also need to be created. Such features are:

   - **Edge Weight:** distance between nodes based on the Euclidean distance (specified by DIMACS dataset)
   - **Global Edge Rank:** rank of the edge compared to other edges in terms of how weightings where 1 denotes the smallest edge weight
   - **Local Edge Rank:** relative rank of incident edges based on the node indices
   - **Node Coordinate u:** The cartesian coordinate of node u of the edge.
   - **Node Coordinate v:** The cartesian coordinate of node v of the edge.

   These are the immediately identifiable features. Other features are found through computation during this task. This task should take a considerable amount of time as there is a large number of edges to iterate through. Furthermore, there is a need to write code to compute the stated features which takes quite some time.

3. **Feature Selection (1 week):**
   The usage of mutual information feature selection technique to rank the features in terms of the information gain, then testing each subset of features up to all features on the model to select the best features to use. The task then involves the selection of a subset of relevant and useful features to construct the model. This task should take the least amount of time compared to others as it is using the function mutual_info_classif and then hyperparameter tuning to achieve the top features to use.

4. **Model Training (3 weeks):**
   It involves the training of multiple different classification models such as Gaussian Naive Bayes, Logistic Regression, Support Vector Machine, and Random Forest using the features selected from the previous step. The classification model will predict whether a given edge is part of the positive or negative class. The goal of the model is to remove any edges predicted negative rather than learning the entire solution. This task will take a considerable amount of time as hyperparameter tuning of the models is involved and also evaluating the performance of selected parameters in order to choose the best one.

5. **Evaluation (3 weeks):**
   The evaluation of this project consists of measuring the duration the solver takes to produce the optimal solution from both the original and pruned instance. The expectation is that the solver runs faster on the pruned instance. Additionally, the optimal solution produced from the pruned instance should be similar to the optimal solution constructed from the original instance. The evaluation of the solver consists of comparing the output produced by the solver with the optimal solution provided by DIMACS. The solver will run on the many instances provided, and the expectation is that the returned solutions match the given optimal solutions. The evaluation of the model consists of measuring the performance of the model using accuracy and recall. The goal is to minimize the false negatives as we want to prune the negative instances with high confidence. The model should have both high accuracy and high recall. Finally, the objective function value is measured before and after the machine learning model prunes the edges. The minimization of the loss in the objective function value occurs if there is a loss. The expectation is this will also take a considerable amount of time as the solver and pruning will be run on many different CVRP sizes, and so certain instances may take days.

6. **Final Report Write Up (2 weeks):**
   In conjunction to continue writing my report as I progress with the tasks, the dedication of the last two weeks is for completing and refining the final report write-up.

7. **Contingency (2 weeks):**
   It is time set aside in the event of emergencies or discrepancies that resulted in work progressing slower than planned.

Appendix B: **Model evaluation charts**

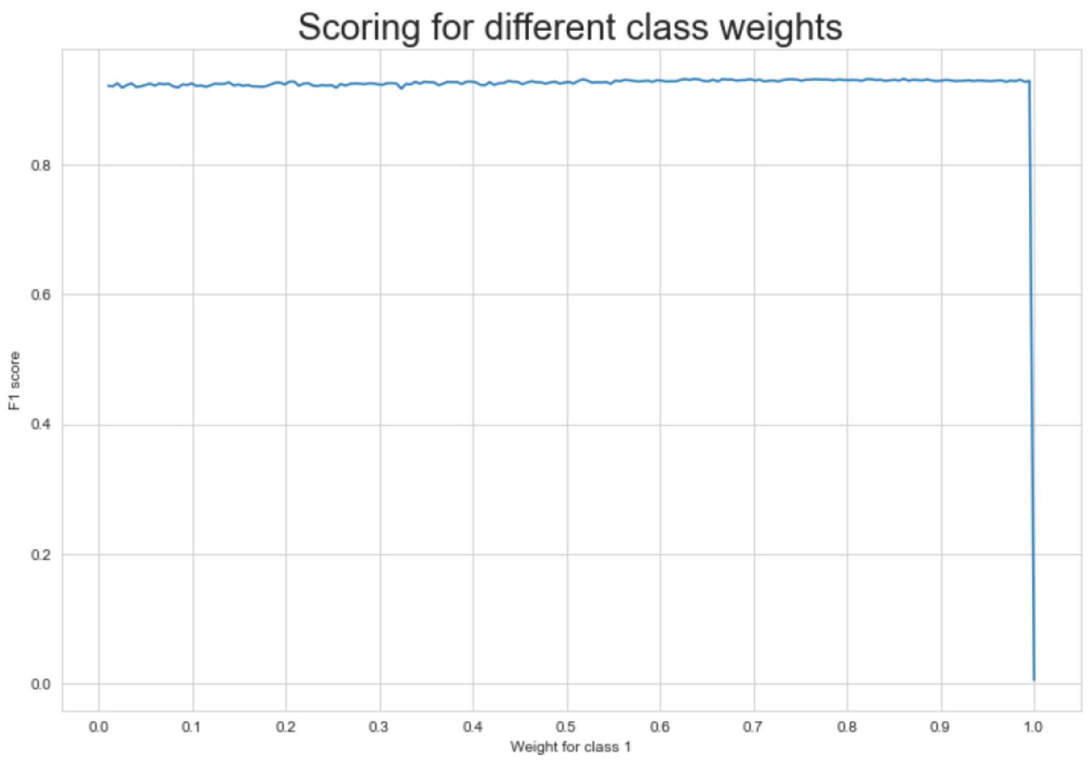


Figure B.1: Optimal class weights using F1 score.

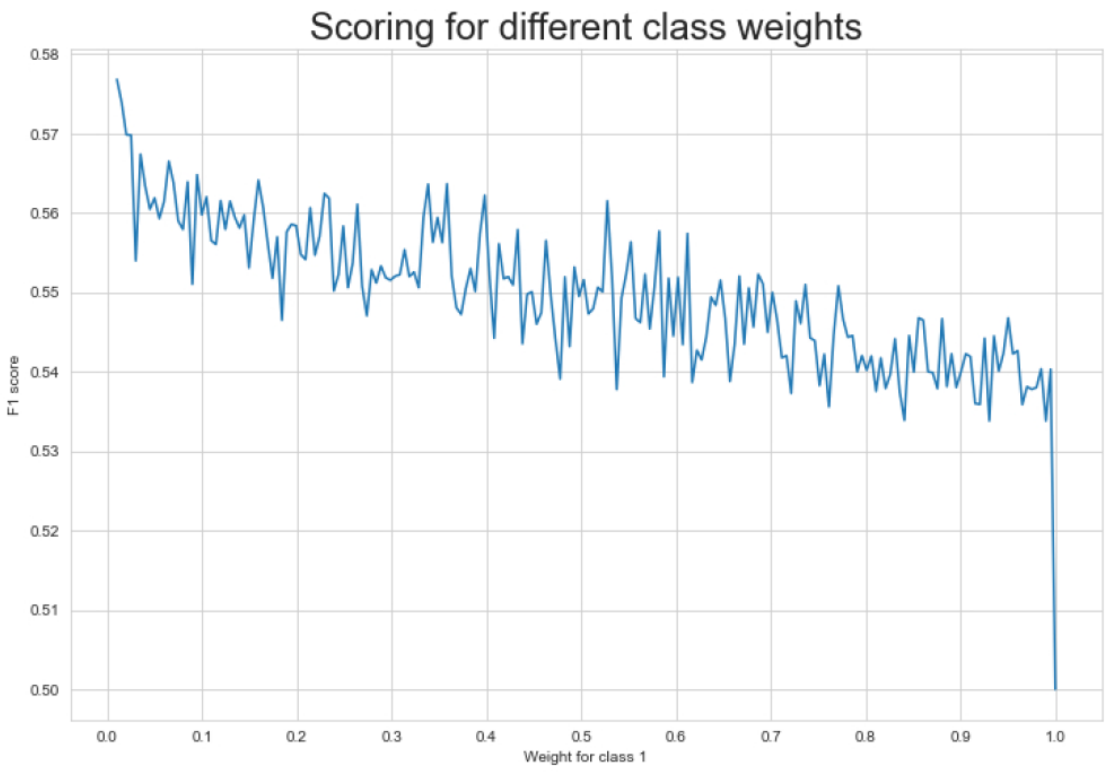


Figure B.2: Class weights.

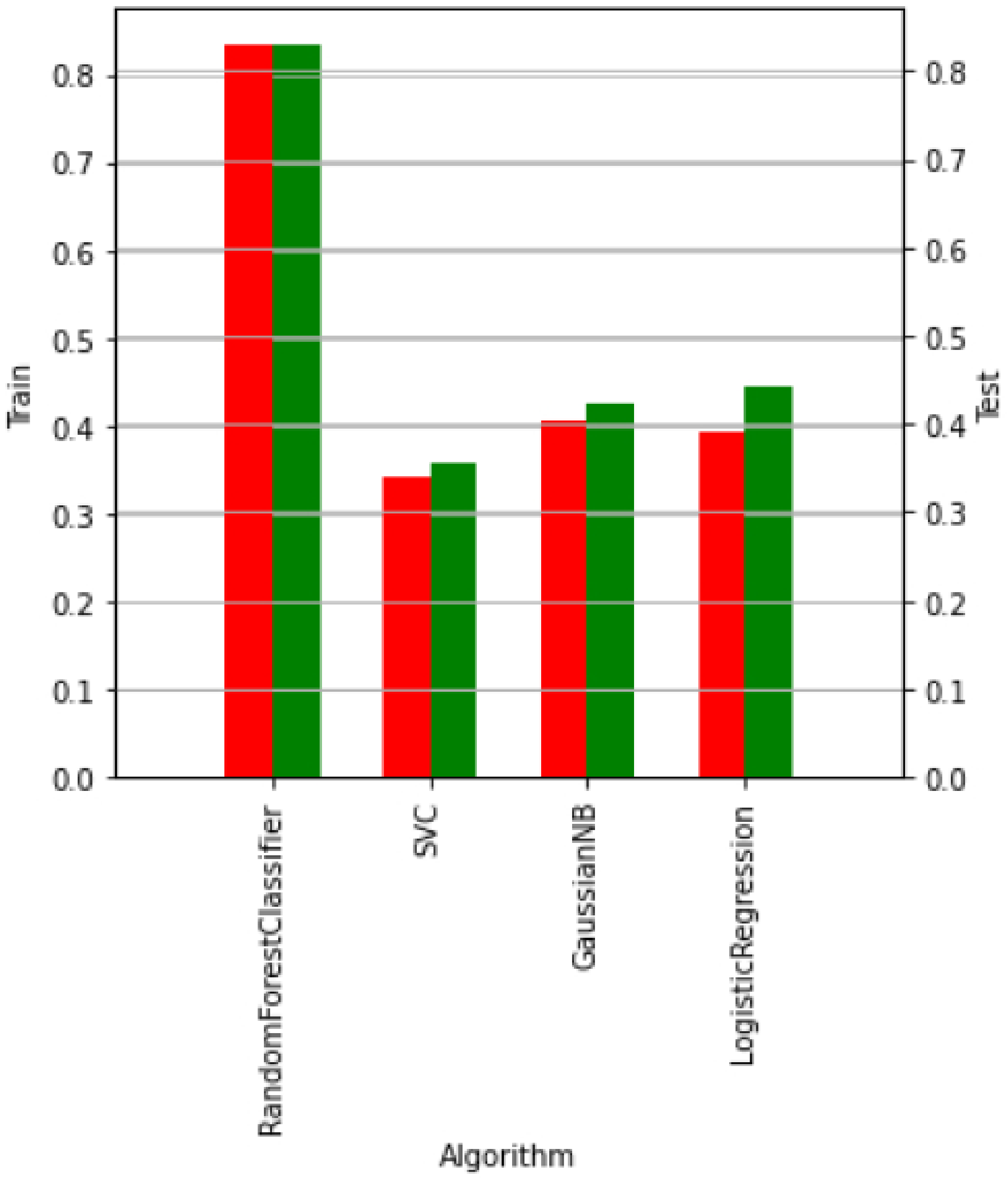


Figure B.3: Train test split after normalisation.

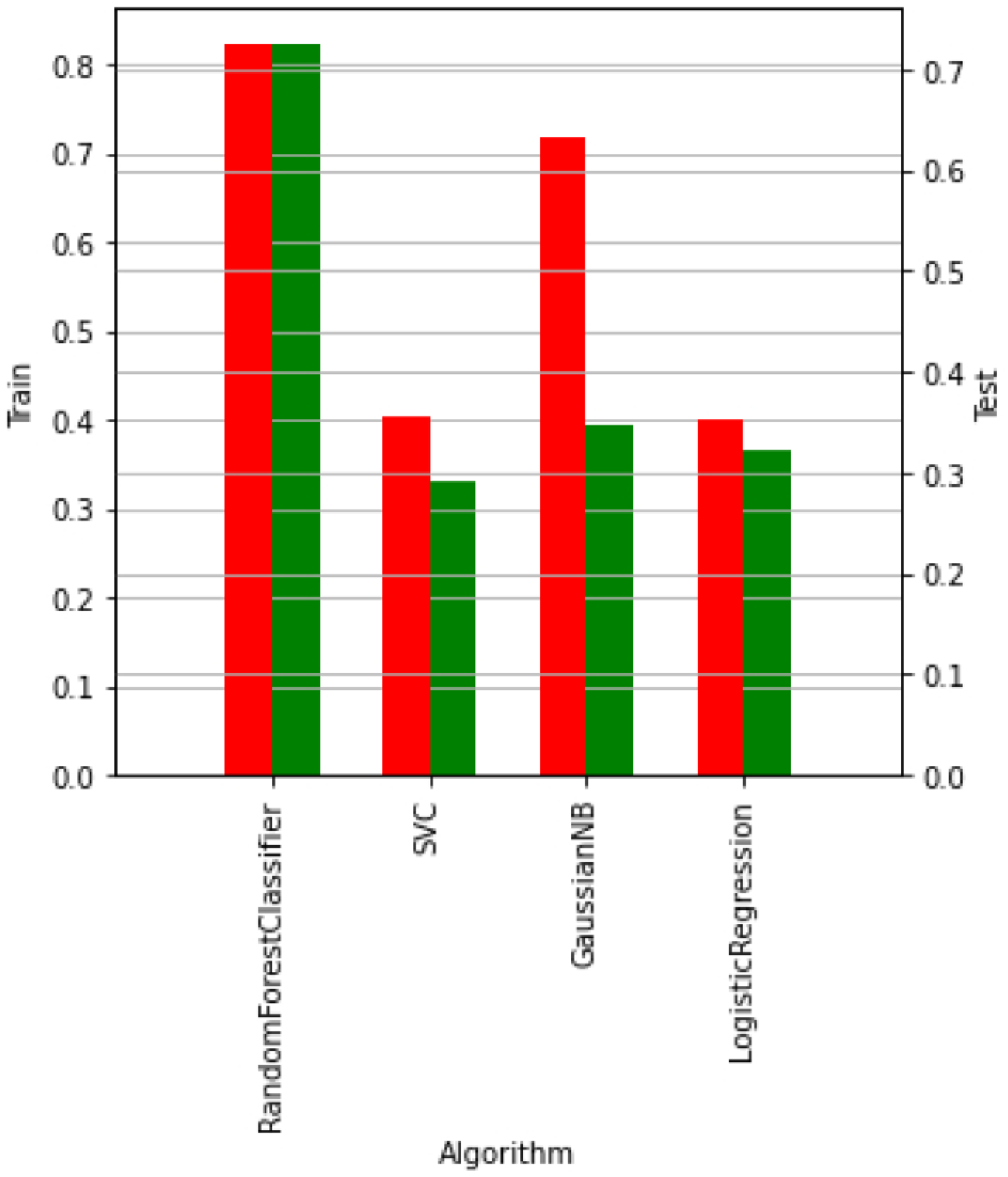


Figure B.4: Train test split before normalisation.

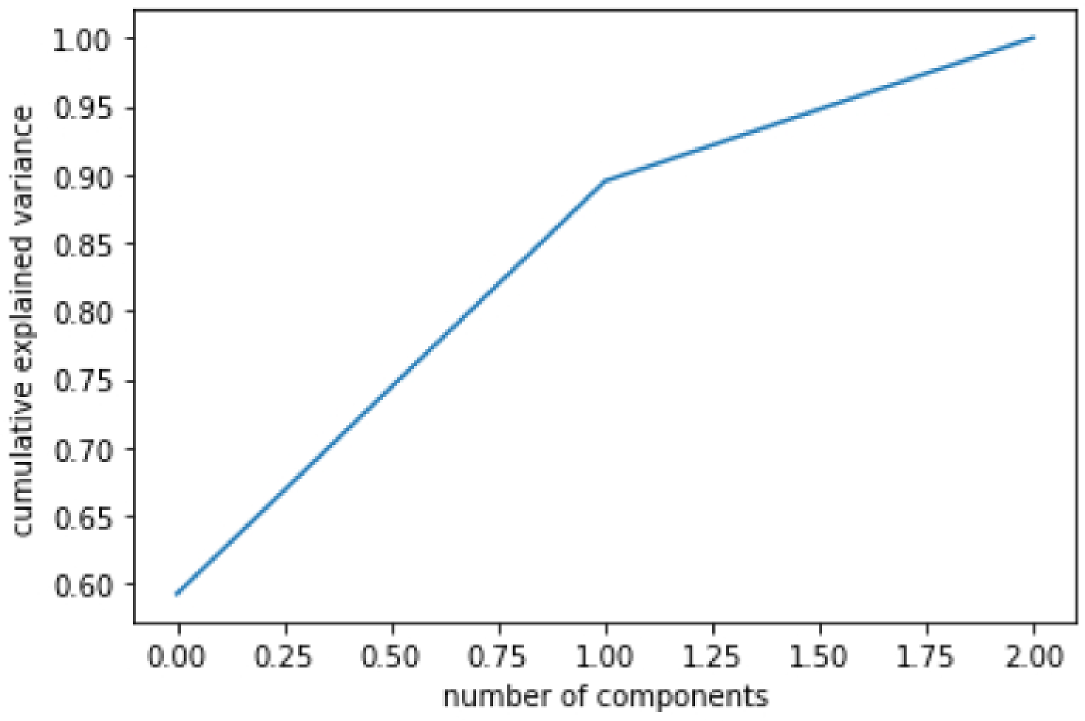


Figure B.5: PCA.

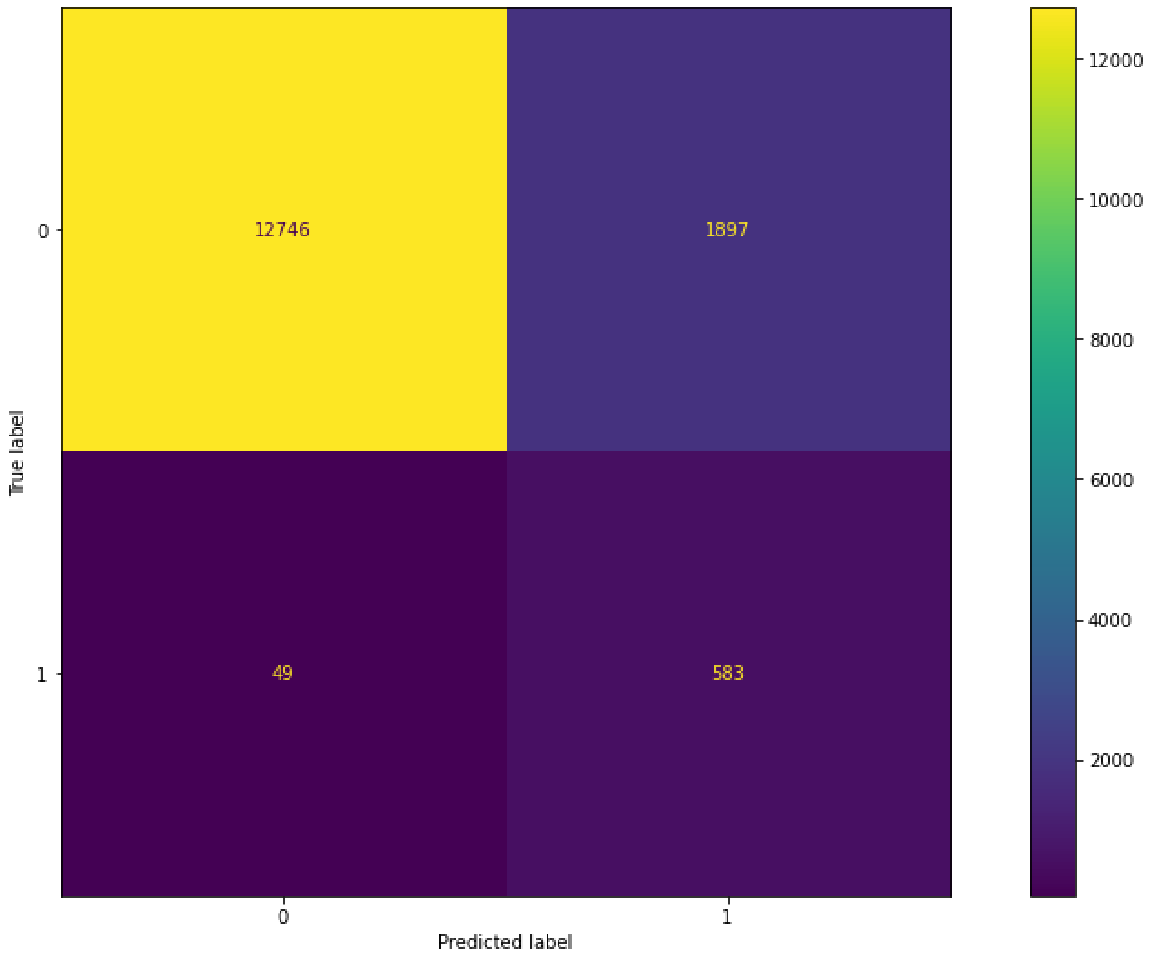


Figure B.6: Logistic Regression Confusion Matrix.

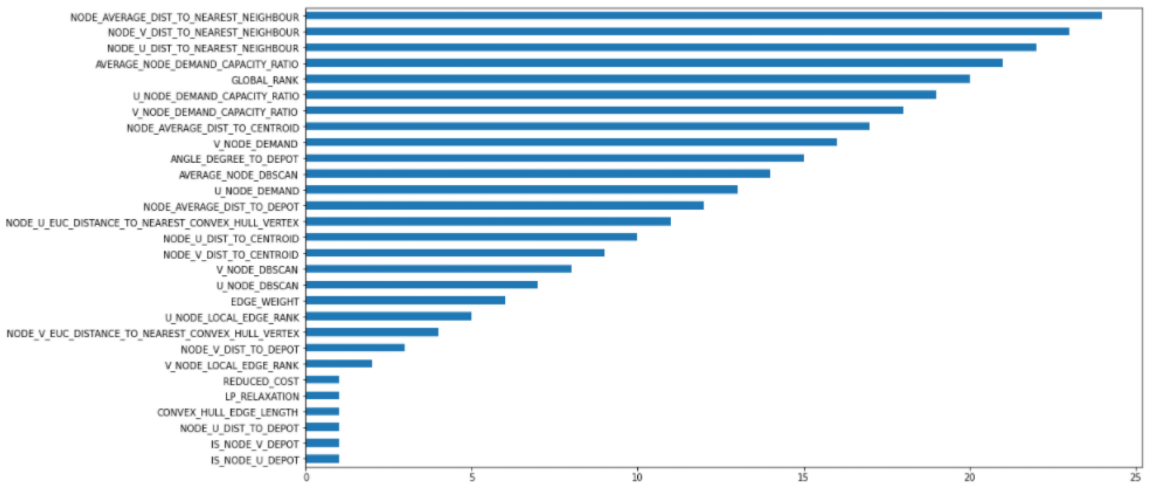


Figure B.7: Logistic Regression Feature Importance.

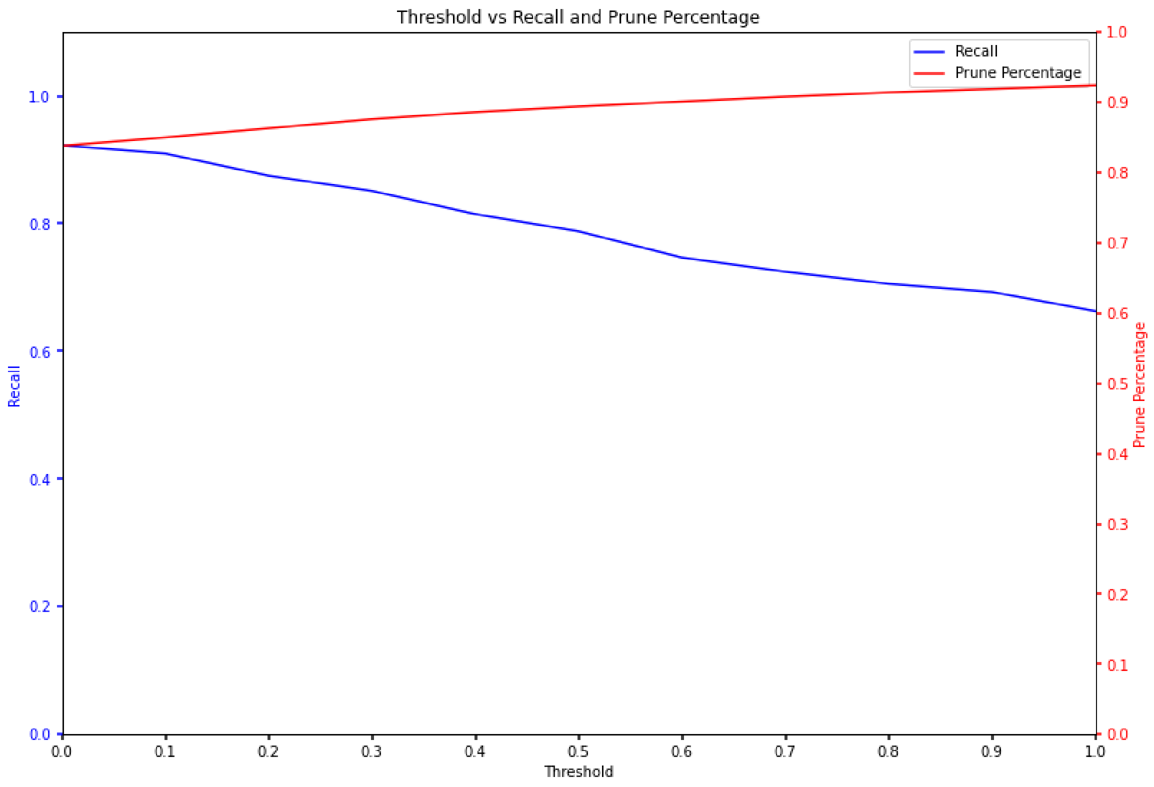


Figure B.8: Logistic Regression Recall and Prune vs Threshold.

```
Optimal number of features : 6
```

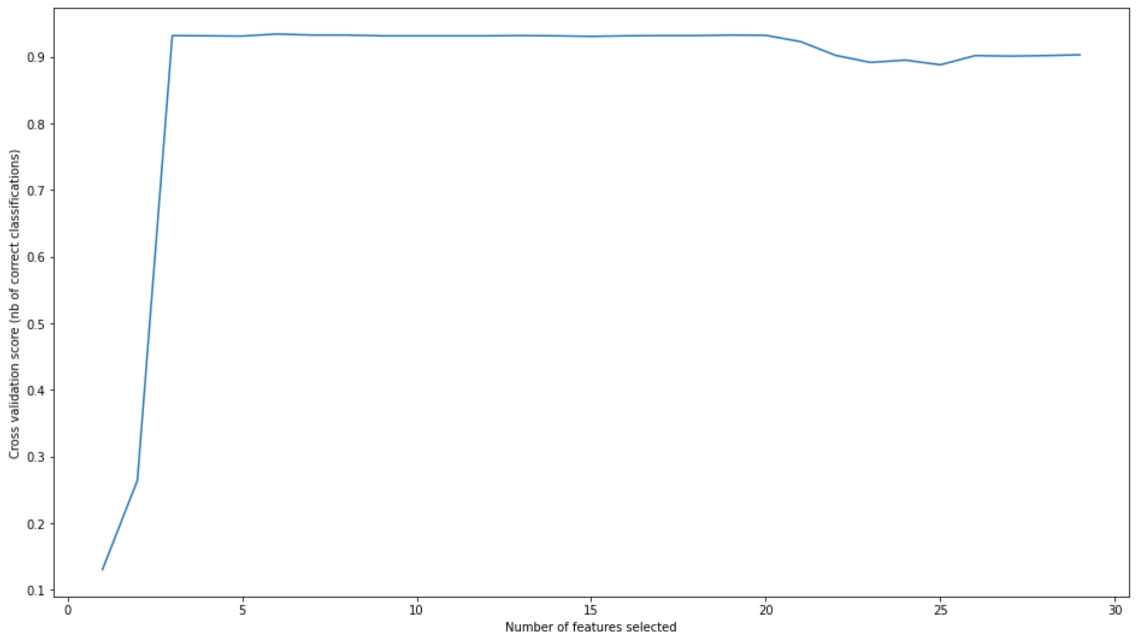


Figure B.9: Logistic Regression RFECV.

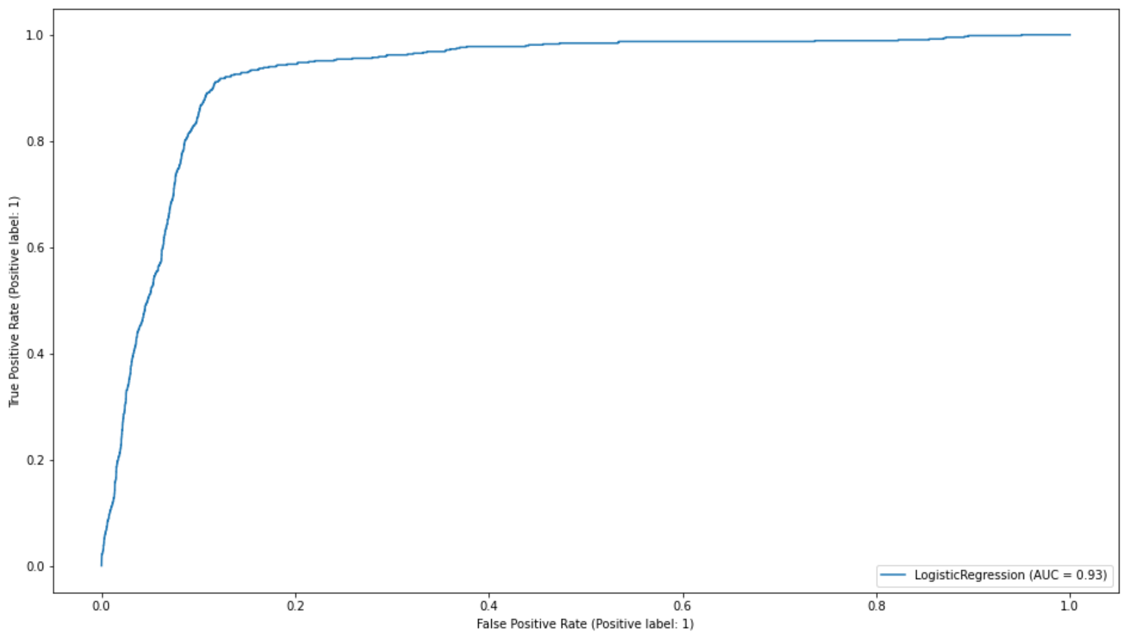


Figure B.10: Logistic Regression ROC.

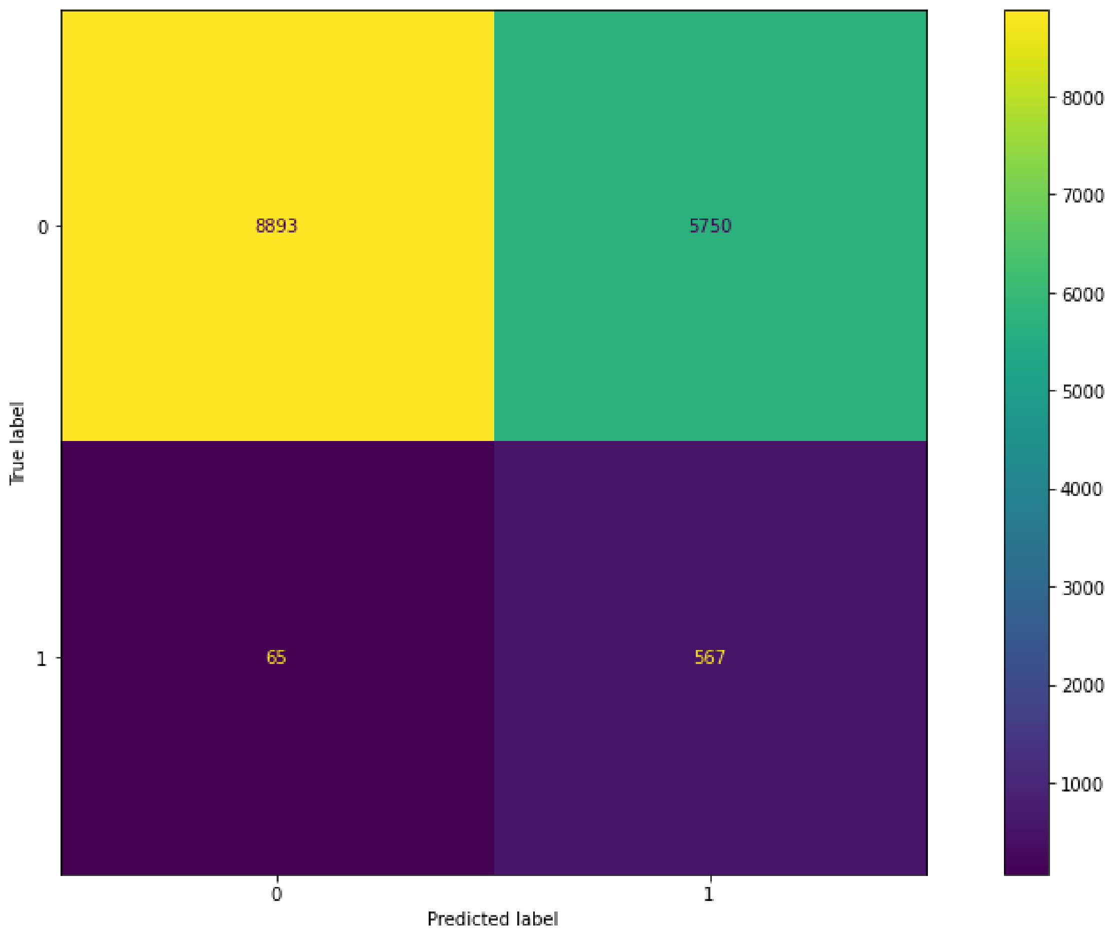


Figure B.11: Logistic Regression Confusion Matrix.

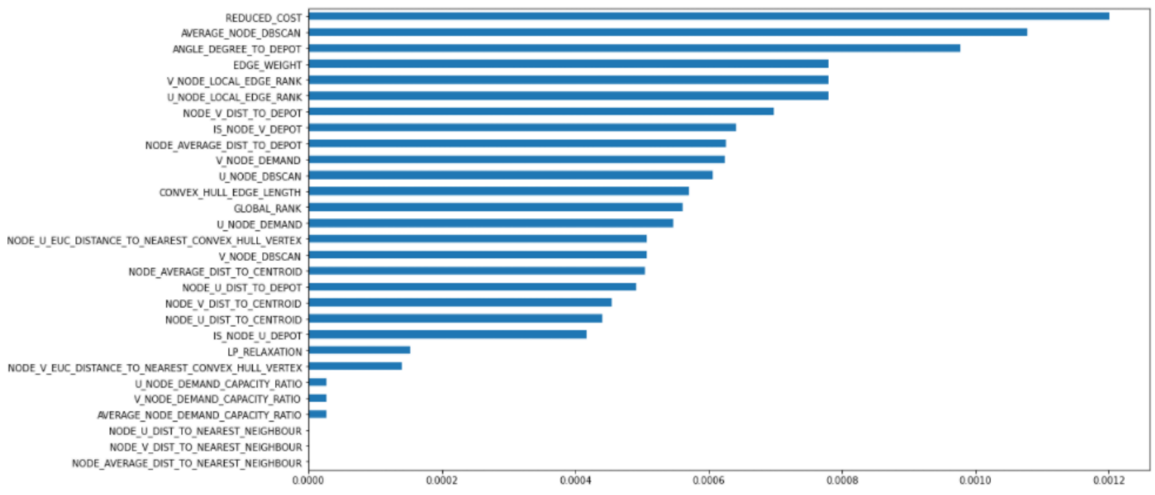


Figure B.12: Naive Bayes Feature Importance.

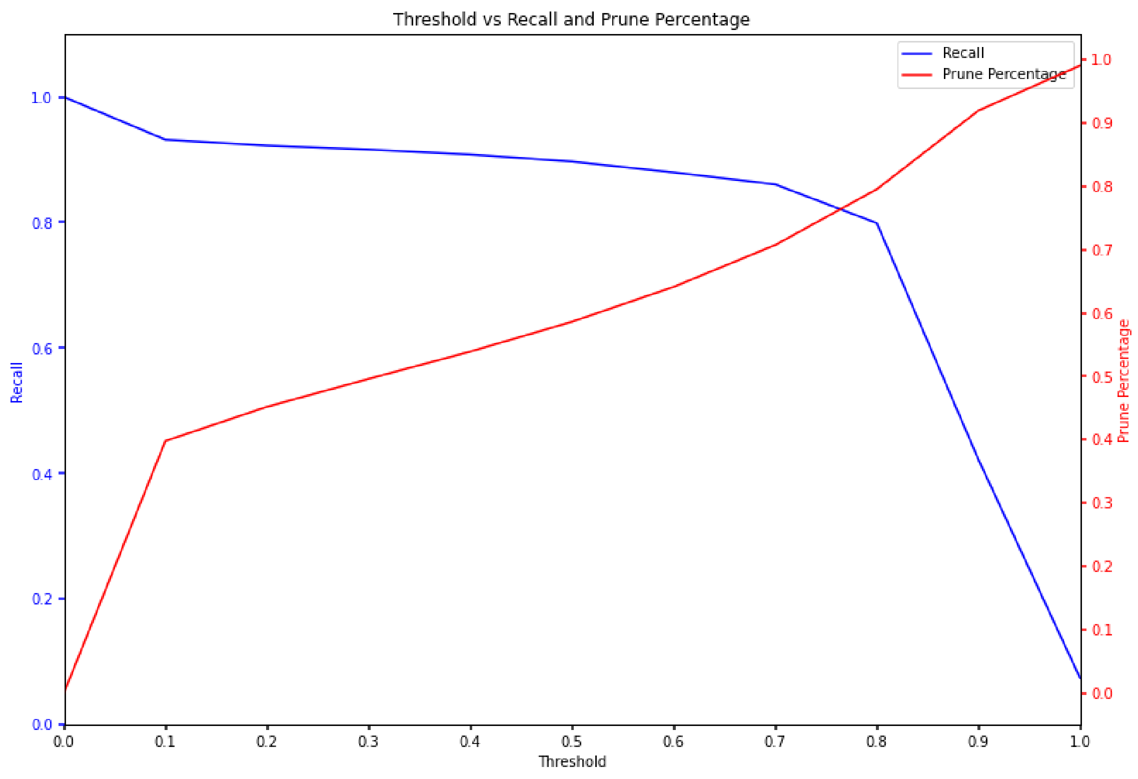


Figure B.13: Naive Bayes Recall and Prune vs Threshold.

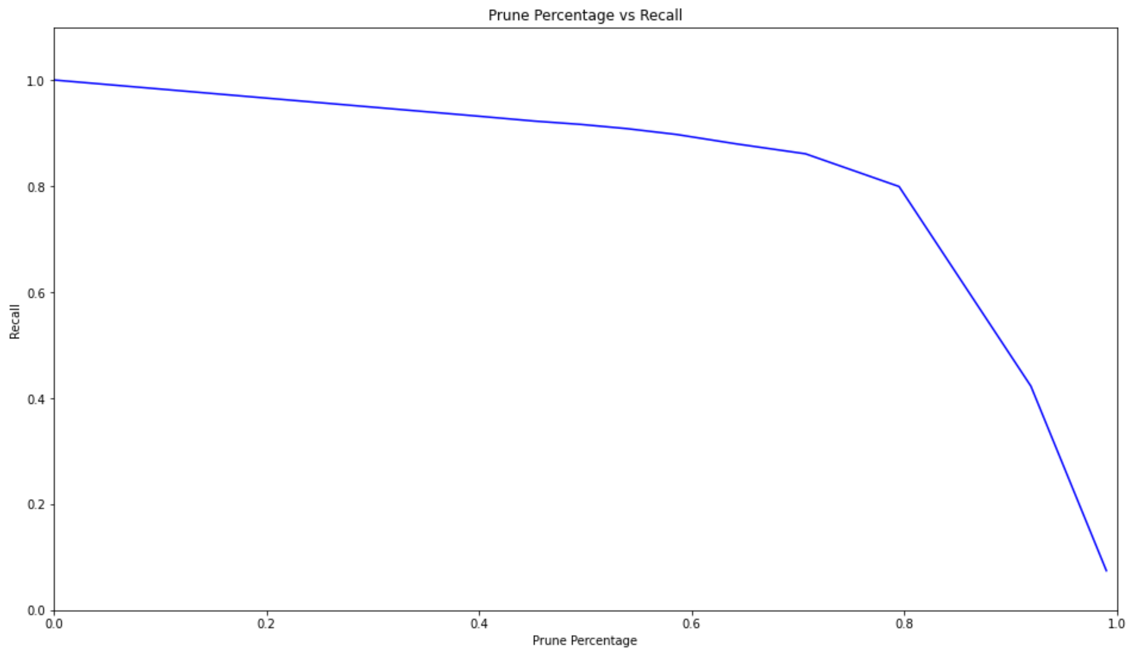


Figure B.14: Naive Bayes Recall vs Prune.

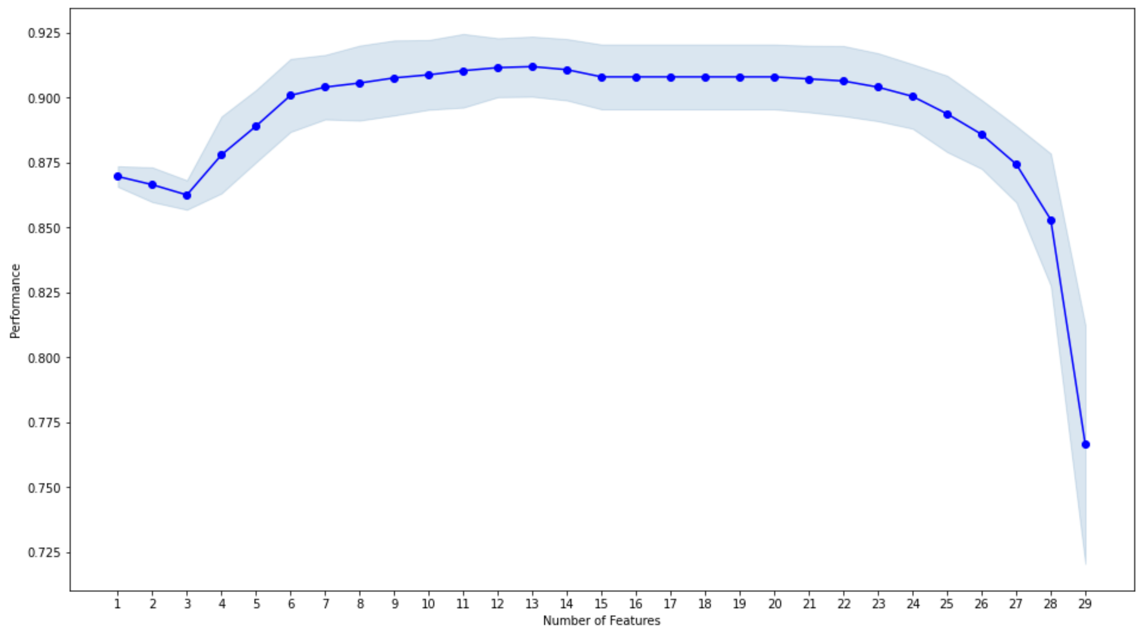


Figure B.15: Naive Bayes RFECV.

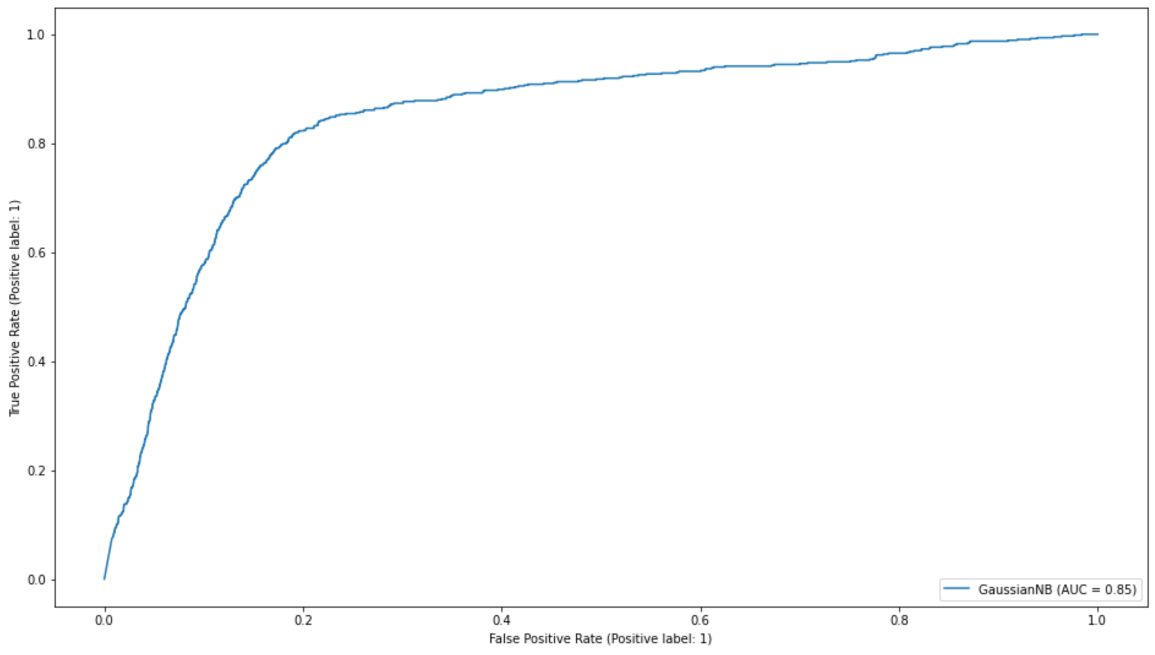


Figure B.16: Naive Bayes ROC.

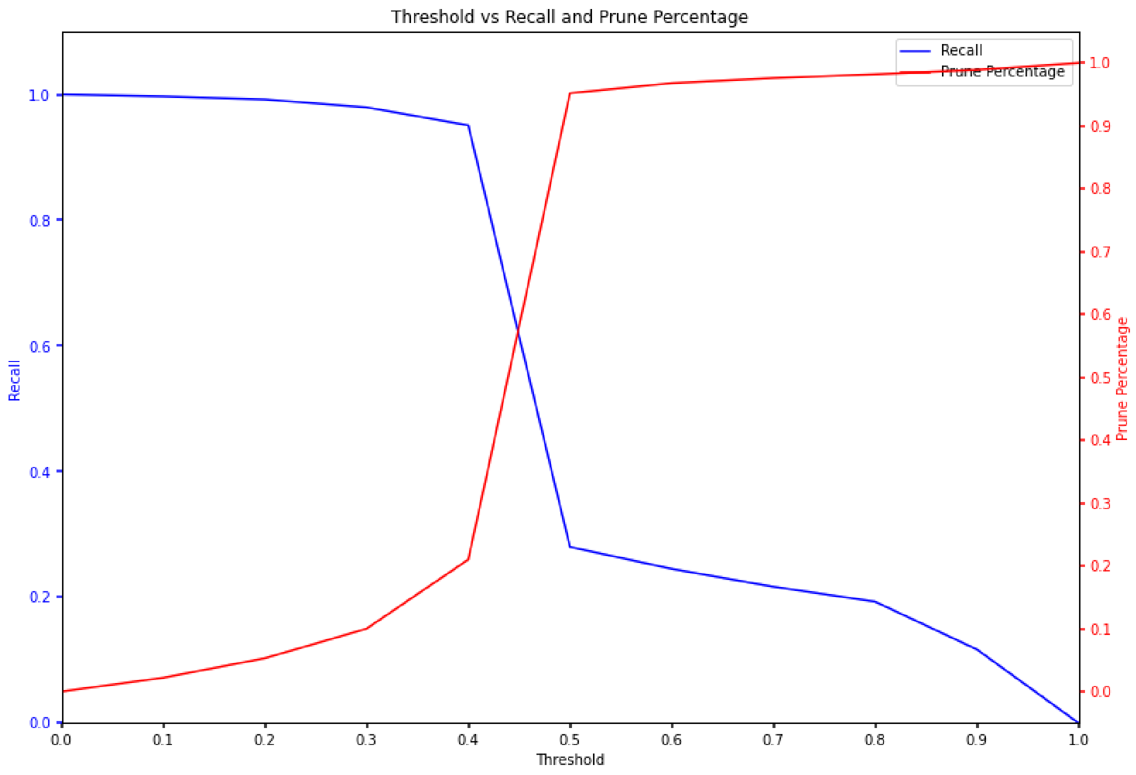


Figure B.17: SVC Recall and Prune vs Threshold.

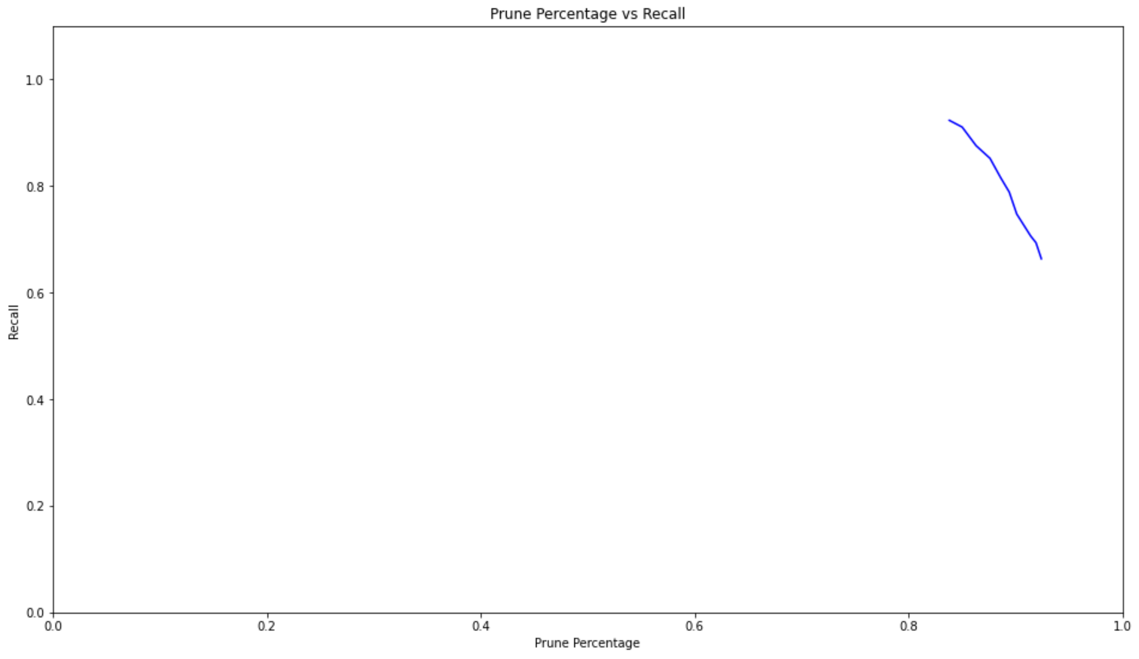


Figure B.18: SVC Recall vs Prune.

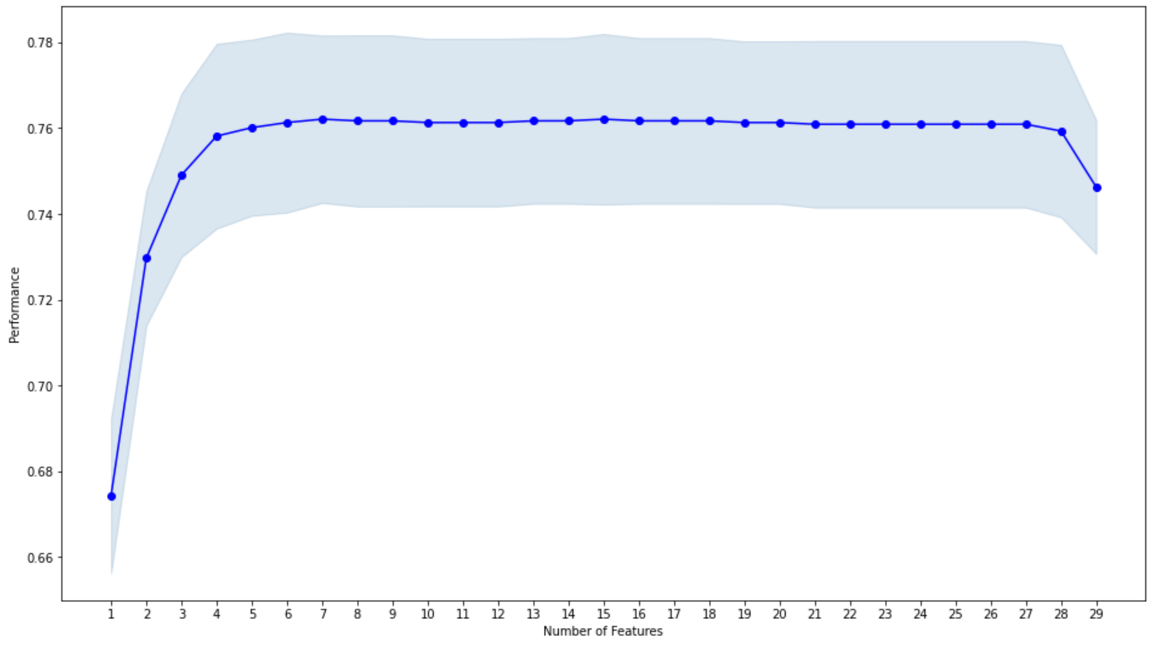


Figure B.19: SVC RFECV.

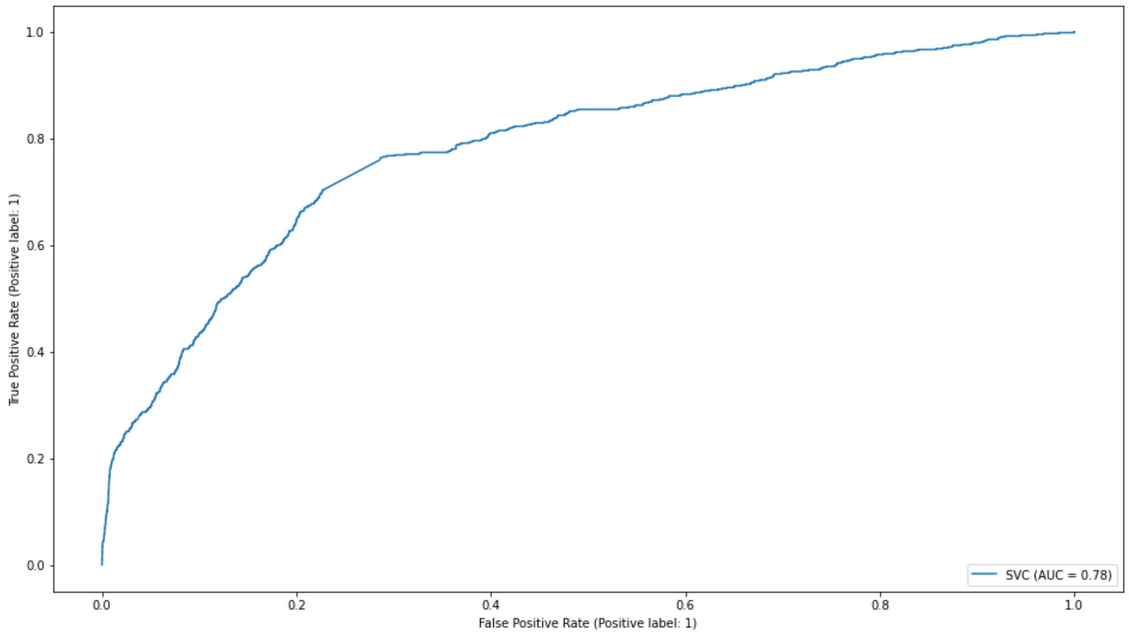


Figure B.20: SVC ROC.

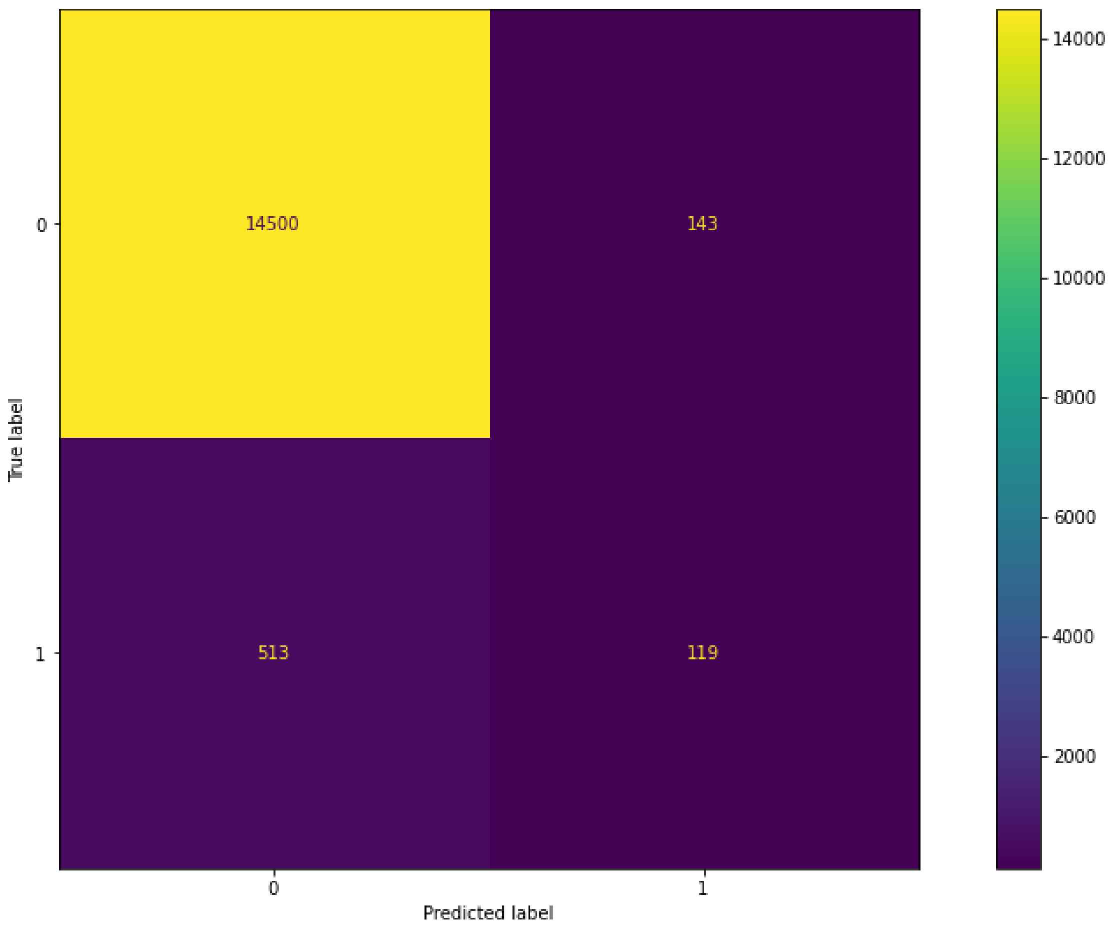


Figure B.21: SVC Confusion Matrix.

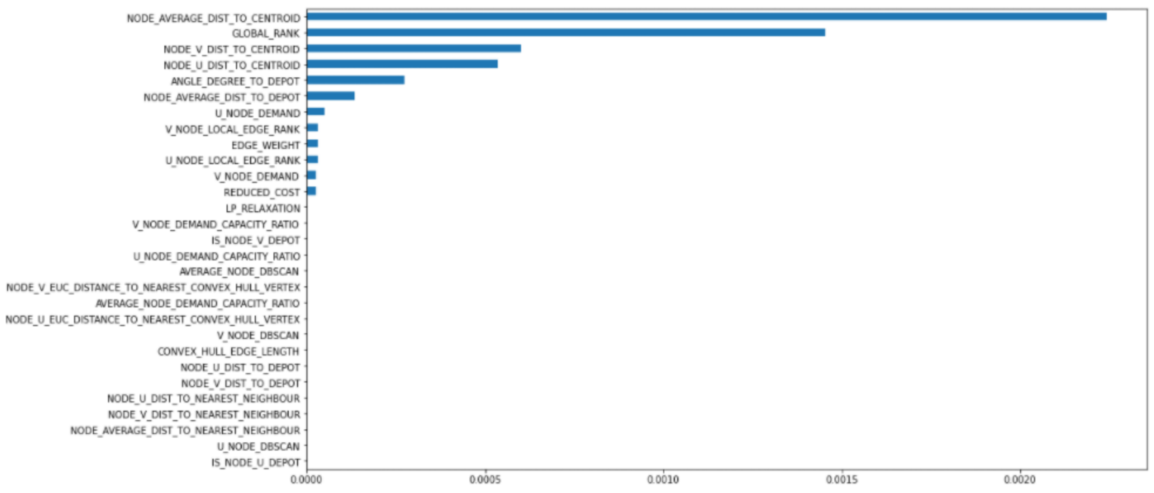


Figure B.22: SVC Feature Importance.

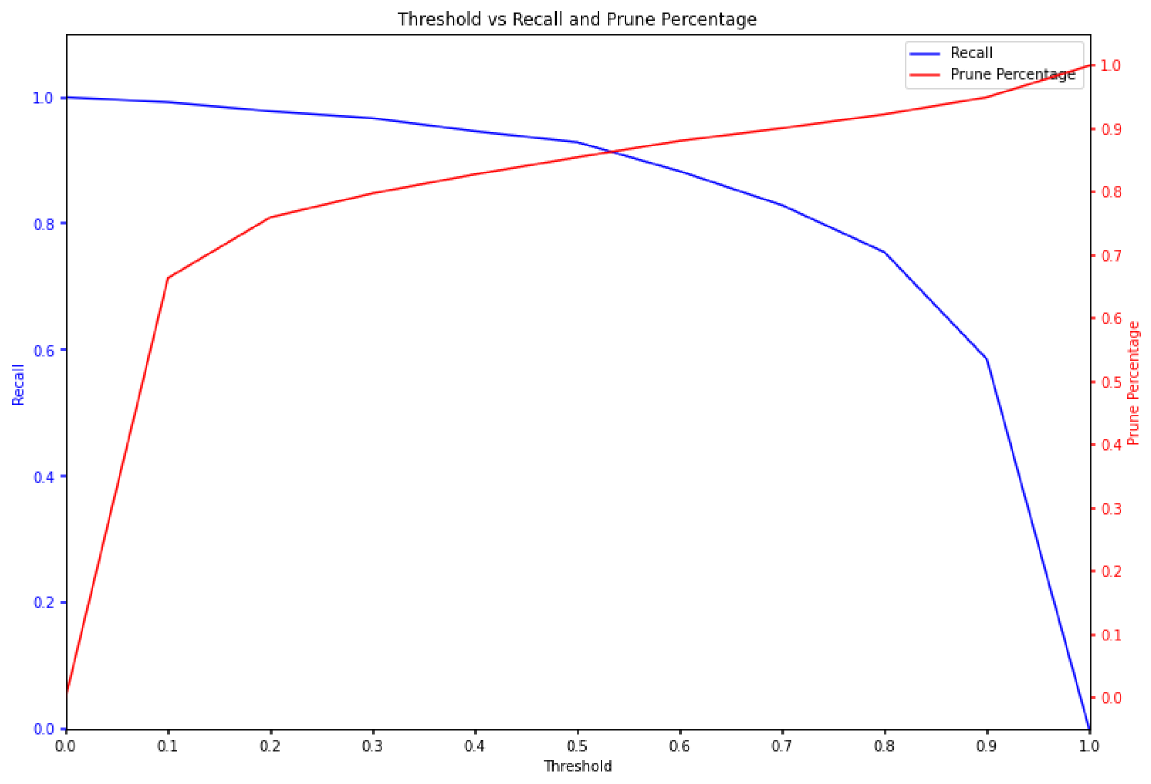


Figure B.23: RFRecall and Prune vs Threshold.

```
Optimal number of features : 23
```

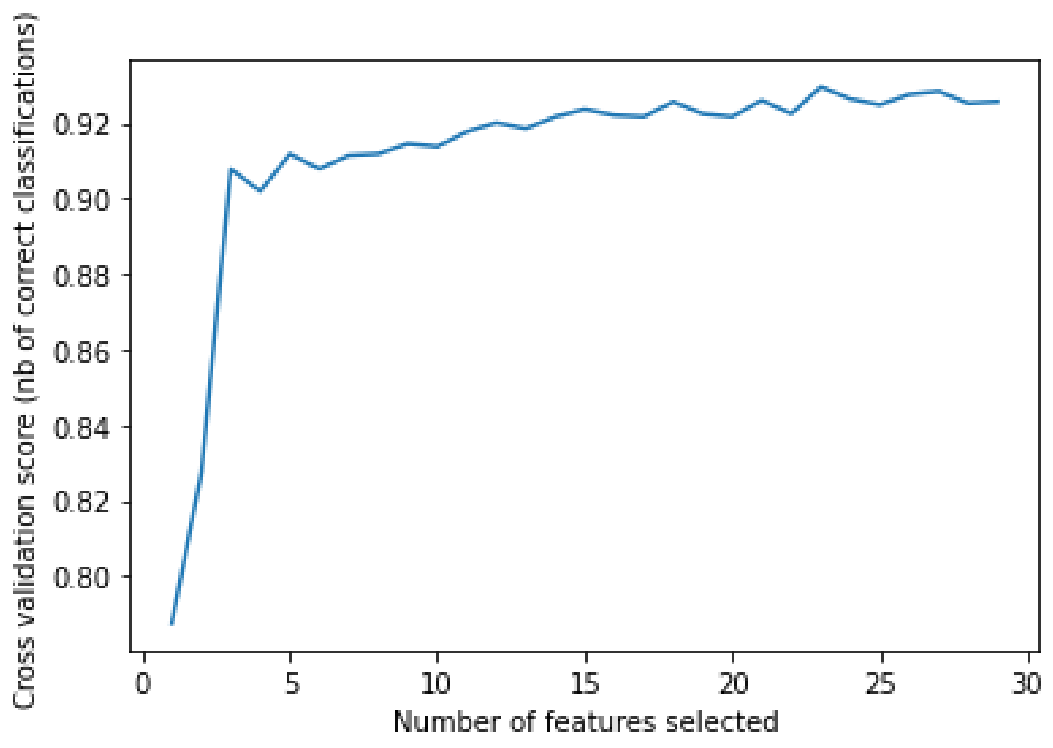


Figure B.24: RF RFECV.

Bibliography

1. Vesselinova, N., Steinert, R., Perez-Ramirez, D. F. & Boman, M. *Learning Combinatorial Optimization on Graphs: A Survey with Applications to Networking* 2020.
2. Toth, P. & Vigo, D. *Models, relaxations and exact approaches for the capacitated vehicle routing problem* (2002), 487–512.
3. Røpke, S. *General rights Heuristic and exact algorithms for vehicle routing problems* ().
4. Baldacci, R., Hadjiconstantinou, E. & Mingozzi, A. An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research* **52.** ISSN: 0030364X (5 2004).
5. Bengio, Y., Lodi, A. & Prouvost, A. *Machine learning for combinatorial optimization: A methodological tour d'horizon* 2020.
6. Lauri, J., Dutta, S., Grassia, M. & Ajwani, D. Learning fine-grained search space pruning and heuristics for combinatorial optimization. http://arxiv.org/abs/2001.01230 (Jan. 2020).
7. Indyk, P. P. *Lecture 1: Intro. Augmenting Bloom filters with ML oracles.* 2019.
8. Kraska, T., Beutel, A., Chi, E. H., Dean, J. & Polyzotis, N. *The case for learned index structures* in (2018).
9. Kraska, P. T. *Lecture 5: Learned Data Structures* 2019.
10. Mitzenmacher, P. M. *Lecture 4: Learned Oracles* 2019.
11. Kumar, R., Purohit, M. & Svitkina, Z. *Improving online algorithms via ML predictions* in. **2018-December** (2018).
12. Purohit, M. *Lecture 7: Improving Online Algorithms with Machine Learning* 2019.
13. Alvarez, A. M., Louveaux, Q. & Wehenkel, L. A Supervised Machine Learning Approach to Variable Branching in Branch-And-Bound. *Ecml* (2014).
14. Baltean-Lugojan, R., Bonami, P., Misener, R. & Tramontani, A. *Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks* (2019).
15. Probst, M., Rothlauf, F. & Grahl, J. Scalability of using Restricted Boltzmann Machines for combinatorial optimization. *European Journal of Operational Research* **256.** ISSN: 03772217 (2 2017).
16. Mazyavkina, N., Sviridov, S., Ivanov, S. & Burnaev, E. *Reinforcement learning for combinatorial optimization: A survey* 2020.
17. Laterre, A. *et al. Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization* 2018.
18. Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* **8.** ISSN: 0885-6125 (3-4 1992).
19. Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B. & Song, L. *Learning combinatorial optimization algorithms over graphs* in. **2017-December** (2017).
20. Trevisan, L. Combinatorial Optimization: Exact and Approximate Algorithms. https://lucatrevisan.github.io/books/cs261.pdf (2011).
21. Vidal, T., Laporte, G. & Matl, P. *A concise guide to existing and emerging vehicle routing problem variants* 2020.

22. Hromkovic, J. *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics* ISBN: 3642079091 (Springer-Verlag, Berlin, Heidelberg, 2010).
23. Christofides, N. *THE VEHBCIE ROUTIWG PROBLEEM (*)* (1976), 55–70.
24. Vidal, T., Crainic, T. G., Gendreau, M. & Prins, C. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research* **234,** 658–673. ISSN: 03772217 (3 May 2014).
25. Cordeau, J.-F., Gendreau, M., Hertz, A., Laporte, G. & Sormany, J.-S. Les Cahiers du GERAD. ISSN: 0711-2440 (2004).
26. Mester, D. & Bräysy, O. Active guided evolution strategies for large scale vehicle routing problem with time windows. *Computers Operations Research* **32,** 1593–1614 (June 2005).
27. Vazirani, V. V. *Approximation Algorithms* (Springer Berlin Heidelberg, 2003).
28. Balinski, M. L. & Quandt, R. E. On an Integer Program for a Delivery Problem. *Operations Research* **12.** ISSN: 0030-364X (2 1964).
29. Baldacci, R., Mingozzi, A. & Roberti, R. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research* **218,** 1–6. ISSN: 03772217 (1 Apr. 2012).
30. Cordeau, J.-F., Laporte, G., Savelsbergh, M. & Vigo, D. in, 195–224 (Jan. 2007).
31. Nazari, M., Oroojlooy, A., Takáč, M. & Snyder, L. V. *Reinforcement learning for solving the vehicle routing problem* in. **2018-December** (2018).
32. Golden, B., Raghavan, S. & Wasil, E. The vehicle routing problem: Latest advances and new challenges. *Operations Research/ Computer Science Interfaces Series* **43.** ISSN: 1387666X (2008).
33. Murata, T. & Itai, R. *Enhancing Solution Similarity in Multi-Objective Vehicle Routing Problems with Different Demand Periods.* ISBN: 9789537619091 (INTECH Open Access Publisher, 2008).
34. Gao, L. *et al.* Learn to Design the Heuristics for Vehicle Routing Problem. http://arxiv.org/abs/2002.08539 (Feb. 2020).
35. Laporte, G. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* **59.** ISSN: 03772217 (3 1992).
36. Benoist, T., Estellon, B., Gardi, F., Megel, R. & Nouioua, K. Localsolver 1. x: a black-box local-search solver for 0-1 programming. *4or* **9,** 299–316 (2011).
37. Gardi, F., Benoist, T., Darlay, J., Estellon, B. & Megel, R. *Mathematical programming solver based on local search* (Wiley Online Library, 2014).
38. Gurobi Optimization, L. *Gurobi Optimizer Reference Manual* 2021. http://www.gurobi.com.
39. Comparisons of Commercial MIP Solvers and an Adaptive Memory (Tabu Search) Procedure for a Class of 0-1 Integer Programming Problems. *Algorithmic Operations Research* **7.** ISSN: 1718-3235 (1 2012).
40. Borčinová, Z. Two models of the capacitated vehicle routing problem. *Croatian Operational Research Review* **8,** 463–469. ISSN: 18489931 (2 2017).
41. P. & Toth, V. An Overview of Vehicle Routing Problems., Monographs on Discrete Mathematics and Applications. In: The Vehicle Routing Problem. *SIAM,* (2000).
42. Arnold, F. & Sörensen, K. What makes a VRP solution good? The generation of problem-specific knowledge for heuristics. *Computers and Operations Research* **106.** ISSN: 03050548 (2019).
43. Arnold, F. *Efficient Heuristics for Routing and Integrated Logistics: Dissertation* ISBN: 9789089941824. https://books.google.ie/books?id=FoowyQEACAAJ (2018).

44. Legrand-Lixon, C. *About the prediction of good edges for the CVRP and its applications* ().

45. Rasku, J., Kärkkäinen, T. & Musliu, N. *Feature extractors for describing vehicle routing problem instances* in. **50** (2016).

46. Vazirani, V. V. *Approximation Algorithms* ISBN: 3642084699 (Springer Publishing Company, Incorporated, 2010).

47. Brilliant.org. Convex Hull. https://brilliant.org/wiki/convex-hull/ (2021).