Εθνικό Μετσόβιο Πολυτεχνείο
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Αλγόριθμοι Καθοδηγούμενοι Από Δεδομένα Για Βελτιστοποίηση Ανάθεσης Πόρων

Διπλωματική Εργασία
Νικόλαος Γουτζούλιας

Επιβλέπων: Δημήτριος Φωτάκης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2024

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Αλγόριθμοι Καθοδηγούμενοι Από Δεδομένα Για Βελτιστοποίηση Ανάθεσης Πόρων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Γουτζούλιας

Επιβλέπων: Δημήτριος Φωτάκης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 09/07/2024.

.....	.....	.....
Δημήτριος Φωτάκης	Αριστείδης Παγουρτζής	Ιωάννης Εμίρης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Κ.Π.Α.

.....

Νικόλαος Γουτζούλιας
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Στην παρούσα διπλωματική εργασία, παρουσιάζουμε μια ολοκληρωμένη λύση για τη βελτιστοποίηση της ανάθεσης πόρων σε μια ασφαλιστική εταιρεία μέσω της ενσωμάτωσης προηγμένων τεχνικών μηχανικής μάθησης με ακέραιο προγραμματισμό. Ξεκινάμε με μια εις βάθος εξερεύνηση διαφόρων μοντέλων μηχανικής μάθησης για την πρόβλεψη βασικών δεικτών απόδοσης (*KPIs*), εστιάζοντας τόσο σε γραμμικά όσο και σε μη γραμμικά μοντέλα παλινδρόμησης. Μεταξύ αυτών, τα μοντέλα δέντρων απόφασης και τυχαίων δασών αποδεικνύονται τα πιο αποτελεσματικά λόγω της ικανότητάς τους να διαχειρίζονται πολύπλοκες, μη γραμμικές σχέσεις δεδομένων. Έτσι, παρέχουν ακριβείς προβλέψεις για τον αριθμό περιστατικών που μπορεί να χειριστεί κάθε συνδυασμός μοτοσυκλετών και πλατφορμών, διευκολύνοντας τις βέλτιστες αποφάσεις ανάπτυξης.

Για την ενίσχυση της λύσης μας, ενσωματώνουμε μια λεπτομερή ανάλυση ακραίων περιπτώσεων και εισάγουμε επιπλέον παραμέτρους όπως η ώρα της ημέρας και ένας προσαρμοσμένος μετρικός δείκτης ¨βαρύτητας¨. Αυτό διασφαλίζει ότι τα μοντέλα μας μπορούν να προσαρμοστούν σε διαφορετικά επιχειρησιακά σενάρια, παρέχοντας αξιόπιστες προβλέψεις υπό ποικίλες συνθήκες. Ενσωματώνοντας αυτές τις προβλέψεις μηχανικής μάθησης σε ένα πλαίσιο ακέραιου προγραμματισμού, επιτρέπεται η βελτιστοποίηση των προγραμμάτων βάρδιας των οδηγών, λαμβάνοντας υπόψη επιχειρησιακούς περιορισμούς όπως τα ωράρια βάρδιας και τις αργίες. Αυτή η προσέγγιση ελαχιστοποιεί τα κόστη ενώ μεγιστοποιεί την αποδοτικότητα των υπηρεσιών.

Επιπλέον, απεικονίζονται τα πλεονεκτήματα της χρήσης της Γλώσσας Προγραμματισμού Βελτιστοποίησης (*OPL*) σε σύγκριση με το ευρέως χρησιμοποιούμενο πλαίσιο *OR – Tools* για την επίλυση σύνθετων προβλημάτων δρομολόγησης οχημάτων και διαχείρισης χωρητικότητας. Μέσα από μια λεπτομερή μελέτη περίπτωσης, δείχνουμε ότι η *OPL* βελτιώνει την ευκρίνεια και την ακρίβεια του μοντέλου ενώ ταυτόχρονα βελτιστοποιεί την απόδοση στην επίλυση προβλημάτων μεγάλης κλίμακας. Αυτή η σύγκριση υπογραμμίζει τη σταθερότητα και την αποτελεσματικότητα του *OPL* στην παροχή υψηλής ποιότητας λύσεων για σύνθετα σενάρια.

Ενδεχόμενη μελλοντική εργασία θα περιλαμβάνει τη βελτίωση αυτών των μοντέλων μηχανικής μάθησης μέσω της εξερεύνησης πιο προηγμένων τεχνικών όπως η βαθιά μάθηση και η ενισχυτική μάθηση για μεγαλύτερη ακρίβεια προβλέψεων και προσαρμοστικότητα. Η συνεχής συνεργασία με βιομηχανικούς εταίρους είναι απαραίτητη για την επικύρωση και τη βελτίωση των μοντέλων σε διάφορα επιχειρησιακά περιβάλλοντα, διασφαλίζοντας τη σταθερότητα και την εφαρμοσιμότητα της λύσης μας.

Λέξεις κλειδιά

Μηχανική Μάθηση, Ακέραιος Προγραμματισμός, Βελτιστοποίηση Ανάθεσης Πόρων, Βασικοί Δείκτες Απόδοσης (KPI), Δέντρα Απόφασης, Τυχαία Δάση, Πρόβλημα Δρομολόγησης Οχημάτων, Διαχείριση Χωρητικότητας, Ενισχυτική Μάθηση, Διασταυρούμενη Επικύρωση, Κανονικοποίηση, Βιομηχανία Ασφάλισης

Abstract

In this thesis, we present a comprehensive solution for optimizing resource allocation in an insurance company by integrating advanced machine learning techniques with integer programming. We start with an in-depth exploration of various machine learning models to predict key performance indicators (KPIs), focusing on both linear and non-linear regression models. Among these, decision trees and random forest models demonstrate the highest effectiveness due to their capability to manage complex, non-linear data relationships. These models provide accurate predictions for the number of incidents that each combination of motorcycles and platforms can handle, facilitating optimized deployment decisions.

To enhance our solution, we incorporate a detailed analysis of edge cases and introduce additional parameters such as time of day and a custom "heaviness" metric. This ensures our models can adapt to diverse operational scenarios, providing reliable predictions under varying conditions. Embedding these machine learning predictions into an integer programming framework allows us to optimize driver shift schedules, taking into account business constraints such as shift lengths and holidays. This approach minimizes costs while maximizing service efficiency, significantly improving operational performance.

Furthermore, we illustrate the advantages of using Optimization Programming Language (OPL) over the commonly used OR-Tools framework for solving complex vehicle routing and capacity management problems. Through a detailed case study, we demonstrate that OPL enhances model clarity and maintainability while optimizing performance in solving large-scale optimization problems. This comparison underscores the effectiveness of OPL in providing high-quality solutions for complex scenarios.

Future work involves refining these machine learning models by exploring more advanced techniques like deep learning and reinforcement learning for greater predictive accuracy and adaptability. Continued collaboration with industry partners is essential to validate and refine the models in various operational environments, ensuring the efficacy and applicability of our solution.

Keywords

Machine Learning, Integer Programming, Resource Allocation Optimization, Key Performance Indicators (KPIs), Regression Models, Decision Trees, Random Forest, Vehicle Routing Problem, Capacity Management, Reinforcement Learning, Cross-Validation, Regularization, Insurance Industry

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον Καθηγητή της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Ε.Μ.Π. κύριο Δημήτρη Φωτάκη για την εμπιστοσύνη, επίβλεψη, βοήθεια, αλλά και την ευεργετική καθοδήγηση που μου προσέφερε στα τελευταία έτη των σπουδών μου. Ευχαριστώ επίσης τον κύριο Δημήτρη Βενιζέλο, διευθύνοντα σύμβουλο της εταιρείας *EmDoT S.A.* για την εμπιστοσύνη που μου έδειξε, την ανάθεση του προβλήματος της εργασίας, για την προσφορά των απαραίτητων υπολογιστικών πόρων και κυρίως την υποστήριξή του. Ευχαριστώ θερμά τον κύριο Θοδωρή Παπαδόπουλο, μηχανικό της *EmDoT* για την συνεργασία, κατανόηση και εξαιρετικά πολύτιμη βοήθειά του καθ'όλη την τελευταία χρονιά. Επιπλέον, θέλω να ευχαριστήσω τους καθηγητές του Ε.Μ.Π. κυρίους Δημήτριο Σούντρη, Εμμανουήλ Ρογδάκη και Ευστάθιο Ζάχο για τις χρήσιμες συμβουλές τους σε ομολογουμένως δύσκολα σημεία κατά την διάρκεια των σπουδών μου. Ακόμα, ευχαριστώ τους γονείς μου για την σταθερή υποστήριξή τους όλα τα χρόνια της ζωής μου, αλλά και για το κίνητρο που μου έδωσαν σε δύσκολες στιγμές. Ευχαριστώ τους φίλους και συμφοιτητές μου με τους οποίους περάσαμε πολλές δοκιμασίες μαζί και θεωρώ πως εξαιτίας αυτού έχουμε καλλιεργήσει μια σχέση που δοκιμάστηκε και διατηρείται. Τέλος, ευχαριστώ τον εαυτό μου που δεν τα παράτησα όλες τις φορές που βρισκόμουν με την ¨πλάτη στον τοίχο¨ και ανυπομονώ να συνεχίσω αυτήν την πορεία στο *Columbia University*.

Νικόλαος Γουτζούλιας,

Αθήνα, 9η Ιουλίου 2024

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εκτεταμμένη Ελληνική Περίληψη

1.1 Εισαγωγή

Η παρούσα διπλωματική εργασία εξετάζει τη χρήση αλγορίθμων μηχανικής μάθησης και προγραμματισμού υπό περιορισμούς για τη βελτιστοποίηση της ανάθεσης πόρων μίας ασφαλιστικής εταιρείας. Ο στόχος της έρευνας είναι η ανάπτυξη μοντέλων που μπορούν να προβλέψουν βασικούς δείκτες απόδοσης (KPI) και η ενσωμάτωσή τους σε ένα πλαίσιο προγραμματισμού περιορισμών για την αποδοτική ανάθεση των βαρδιών των οδηγών.

Στο κεφάλαιο 6, εξετάζεται η ολιστική προσέγγιση για τη βελτιστοποίηση των μοντέλων. Αυτή περιλαμβάνει διάφορες παραμέτρους όπως ο συνολικός αριθμός περιστατικών, η ώρα της ημέρας και η κυκλοφορική συμφόρηση. Η πολυπλοκότητα των πραγματικών δεδομένων και η εξασφάλιση της γενικευσιμότητας των μοντέλων αποτελούν σημαντικές προκλήσεις που αντιμετωπίζονται με την κατάλληλη κανονικοποίηση των δεδομένων και τη χρήση τεχνικών εξάλειψης χαρακτηριστικών για την αποφυγή υπερπροσαρμογής.

Στο πειραματικό μέρος της εργασίας, χρησιμοποιήθηκαν διάφοροι αλγόριθμοι μηχανικής μάθησης όπως η γραμμική παλινδρόμηση, τα δέντρα αποφάσεων, τα τυχαία δάση και τα νευρωνικά δίκτυα. Μέσω συγκριτικής ανάλυσης, διαπιστώθηκε ότι τα τυχαία δάση και τα νευρωνικά δίκτυα ήταν τα πιο αποδοτικά για την πρόβλεψη των KPI. Η διαδικασία εκπαίδευσης των μοντέλων πραγματοποιήθηκε με τη χρήση του λογισμικού *scikit − learn*, το οποίο προσφέρει πληθώρα εργαλείων και μεθόδων για την ανάλυση δεδομένων και την ανάπτυξη μοντέλων.

Η ανάλυση των αποτελεσμάτων έδειξε σημαντική βελτίωση στην ποιότητα των προβλέψεων των μοντέλων με την ολιστική προσέγγιση. Η προσθήκη παραμέτρων όπως ο αριθμός των περιστατικών και η ώρα της ημέρας ενίσχυσε την ακρίβεια και τη γενικευσιμότητα των μοντέλων. Ωστόσο, ιδιαίτερη προσοχή δόθηκε στην αποφυγή της υπερπροσαρμογής, χρησιμοποιώντας τεχνικές όπως η *cross − validation* και η κανονικοποίηση των δεδομένων.

Συνοψίζοντας, η έρευνα αυτή δείχνει ότι η συνδυασμένη χρήση μηχανικής μάθησης και προγραμματισμού περιορισμών μπορεί να βελτιώσει σημαντικά την αποδοτικότητα μιας ασφαλιστικής εταιρείας στην ανάθεση πόρων. Η ενσωμάτωση προηγμένων τεχνικών μηχανικής μάθησης και η ολιστική προσέγγιση στην εκπαίδευση των μοντέλων

αποτελούν κρίσιμους παράγοντες για την επίτευξη των στόχων της εταιρείας.

1.2 Ανασκόπηση της Βιβλιογραφίας

Η χρήση της μηχανικής μάθησης για την πρόβλεψη βασικών δεικτών απόδοσης (KPI) αποτελεί ένα ιδιαίτερα δημοφιλές επιστημονικό πεδίο που έχει μελετηθεί εκτενώς. Οι τεχνικές που χρησιμοποιούνται σε αυτή τη διπλωματική εργασία έχουν αναπτυχθεί από επιστήμονες που εξετάζουν εφαρμογές της μηχανικής μάθησης. Ένα χαρακτηριστικό παράδειγμα είναι η μελέτη για την ακρίβεια της πρόβλεψης KPI χρησιμοποιώντας τεχνικές μηχανικής μάθησης στον τομέα της αυτοκινητοβιομηχανίας .

Το ζήτημα της πρόβλεψης που σχετίζεται με KPI αντιμετωπίστηκε από τον *Manenti* το 2009, ο οποίος επίσης παρείχε παραδείγματα για το πώς η παρακολούθηση της απόδοσης των διαδικασιών μεταβαίνει γρήγορα από τις αντιδραστικές στις προγνωστικές προσεγγίσεις . Σε μια μελέτη του 2011, οι *Solomon* και *Litoiu* διερεύνησαν έναν νέο τρόπο δημιουργίας ενός δυναμικού προγνωστικού μοντέλου που βοηθά στη λήψη πιο αποτελεσματικών αποφάσεων βελτιστοποίησης επιχειρησιακών διαδικασιών μέσω ενός μοντέλου προσομοίωσης που καθορίζει μια σχέση μεταξύ των μεταβλητών της διαδικασίας και των KPI .

Οι σύγχρονες τεχνικές μηχανικής μάθησης, όπως οι τυχαία δάση και τα νευρωνικά δίκτυα, έχουν αποδειχθεί ιδιαίτερα αποδοτικές στην πρόβλεψη των KPI. Σε έρευνες που αφορούν την αποδοτικότητα του εξοπλισμού, οι μέθοδοι αυτές έχουν χρησιμοποιηθεί για να προβλέψουν την αποτελεσματικότητα και να εντοπίσουν τις κύριες απώλειες που μειώνουν την αποδοτικότητα των μηχανών . Η μελέτη του *El Mazgualdi* έδειξε ότι τα πιο αποδοτικά μοντέλα περιλαμβάνουν αλγορίθμους που αποτελούνται από τεχνικές μάθησης σύνολου και νευρωνικών δικτύων .

Στην παρούσα εργασία, χρησιμοποιούνται διάφορα μοντέλα μηχανικής μάθησης, όπως η γραμμική παλινδρόμηση, τα δέντρα αποφάσεων και τα τυχαία δάση, για την πρόβλεψη των KPI. Η ανάλυση των μοντέλων αυτών δείχνει ότι τα τυχαία δάση και τα νευρωνικά δίκτυα είναι τα πιο αποδοτικά για τις ανάγκες της ασφαλιστικής εταιρείας. Η διαδικασία εκπαίδευσης των μοντέλων πραγματοποιείται με τη χρήση του λογισμικού *scikit−learn*, το οποίο προσφέρει πληθώρα εργαλείων και μεθόδων για την ανάλυση δεδομένων και την ανάπτυξη μοντέλων .

Επιπλέον, η ενσωμάτωση προηγμένων τεχνικών όπως η κανονικοποίηση, η εξάλειψη χαρακτηριστικών και η διασταυρούμενη επικύρωση συμβάλλει στην αποφυγή της υπερπροσαρμογής και στην εξασφάλιση της γενικευσιμότητας των μοντέλων. Αυτή η ολιστική προσέγγιση βελτιώνει την ακρίβεια και την αξιοπιστία των προβλέψεων, καθιστώντας τα μοντέλα πιο αποδοτικά και αποτελεσματικά στις πραγματικές εφαρμογές.

Με τη συστηματική ανάλυση και επιλογή των πιο σχετικών χαρακτηριστικών, η έρευνα αυτή ενισχύει τη δύναμη πρόβλεψης των μοντέλων και μειώνει την υπολογιστική πολυπλοκότητα. Η ενσωμάτωση των προβλέψεων των μοντέλων μηχανικής μάθησης σε ένα πλαίσιο προγραμματισμού περιορισμών προσφέρει ένα ισχυρό εργαλείο για την αποδοτική διαχείριση των πόρων και την επίτευξη των στόχων της ασφαλιστικής εταιρείας.

1.3 Βασικές Αρχές της Μηχανικής Μάθησης

Η μηχανική μάθηση αναφέρεται στην ανάπτυξη στατιστικών αλγορίθμων που έχουν στόχο την εκτέλεση εργασιών χωρίς ρητό προγραμματισμό, μαθαίνοντας από γνωστά δεδομένα και γενικεύοντας σε άγνωστα, έτσι ώστε να γίνονται υπολογισμένες προβλέψεις. Ο όρος μηχανική μάθηση συνδέεται συχνά με τον *Dr. Frank Rosenblatt* από το Πανεπιστήμιο *Cornell*, ο οποίος ανέπτυξε μια μηχανή που μπορούσε να αναγνωρίζει γράμματα της αλφαβήτου. Από τότε, τα μοντέλα μηχανικής μάθησης έχουν προσελκύσει τεράστιο ενδιαφέρον από ακαδημαϊκούς και επαγγελματίες, όντας στο χείλος της μεταμόρφωσης των λειτουργικών τους παραδειγμάτων. Σε αυτό το κεφάλαιο, θα παρουσιάσουμε μερικά από τα βασικά μοντέλα μηχανικής μάθησης και διάφορους τρόπους με τους οποίους μπορούν να χρησιμοποιηθούν για την αντιμετώπιση σύνθετων προβλημάτων κατανομής πόρων.

1.3.1 Γραμμικά Μοντέλα Παλινδρόμησης

Η γραμμική παλινδρόμηση είναι μία από τις απλούστερες μορφές επιβλεπόμενης μάθησης, και αποτελεί ένα χρήσιμο εργαλείο για την πρόβλεψη ποσοτικών απαντήσεων. Είναι μια ευρέως χρησιμοποιούμενη στατιστική μέθοδος που διαχωρίζει τις μεταβλητές του προβλήματος σε εξαρτημένες και μία ή περισσότερες ανεξάρτητες. Η βασική ιδέα είναι να προσαρμοστεί μια γραμμική εξίσωση στα παρατηρούμενα δεδομένα, η οποία μπορεί στη συνέχεια να χρησιμοποιηθεί για να προβλέψει αποτελέσματα για νέα, άγνωστα δεδομένα.

Απλή Γραμμική Παλινδρόμηση: Περιλαμβάνει μία εξαρτημένη μεταβλητή Y και έναν μόνο X, με τη σχέση μεταξύ των δύο να προσεγγίζεται γραμμικά.

$$Y = b_0 + b_1 X + \epsilon \tag{1.1}$$

Πολλαπλή Γραμμική Παλινδρόμηση: Περιλαμβάνει περισσότερους από έναν X, καθιστώντας το μοντέλο πιο ικανό να αντιμετωπίσει προβλήματα του πραγματικού κόσμου όπου η απόδοση εξαρτάται από πολλαπλές μεταβλητές.

$$Y = b_0 + b_1 X_1 + b_2 X_2 + \cdots + \beta_n X_n + \epsilon \tag{1.2}$$

1.3.2 Μη Γραμμικά Μοντέλα Παλινδρόμησης

Τα μη γραμμικά μοντέλα παλινδρόμησης είναι ισχυρά εργαλεία για την κατανόηση και πρόβλεψη αποτελεσμάτων όπου η σχέση μεταξύ των ανεξάρτητων και εξαρτημένων μεταβλητών είναι σύνθετη και δεν ακολουθεί ευθεία γραμμή. Οι ικανότητές τους να παράγουν κυρτές προβλέψεις τους καθιστούν ιδιαίτερα προσαρμόσιμους σε ένα ευρύ φάσμα δομών δεδομένων.

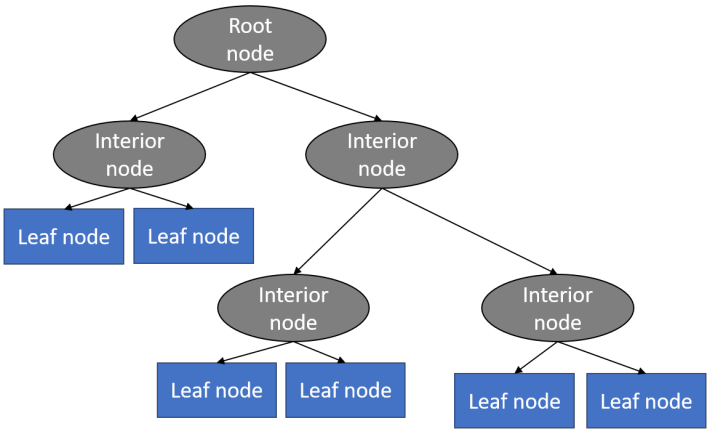
Πολυωνυμική Παλινδρόμηση: Επεκτείνει τη γραμμική παλινδρόμηση προσθέτοντας όρους υψηλότερης τάξης για να αντιμετωπίσει μη γραμμικές σχέσεις.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \cdots + \beta_{n1} X_1^n + \beta_{n+1} X_2 + \beta_{n+2} X_2^2 + \cdots + \beta_m X_2^n + \cdots + \beta_k X_p^n + \epsilon \tag{1.3}$$

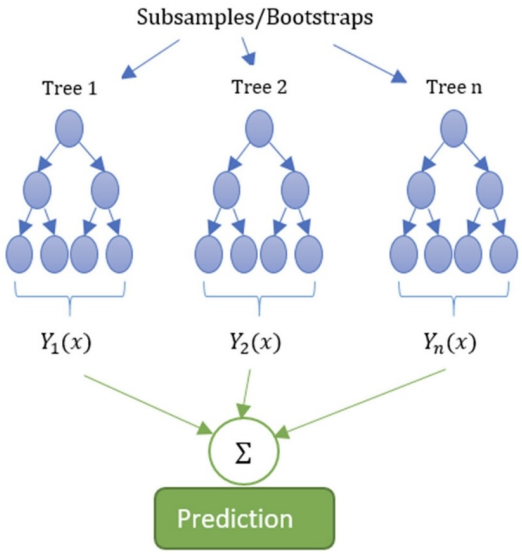
k – Nearest Neighbors: Μη παραμετρική μέθοδος ταξινόμησης που χρησιμοποιεί την απόσταση μεταξύ των σημείων δεδομένων για να προβλέψει την κατηγορία ενός νέου σημείου.

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i \tag{1.4}$$

Δέντρα Απόφασης και Τυχαία Δάση: Τα δέντρα απόφασης χρησιμοποιούν μια ιεραρχική προσέγγιση για την κατηγοριοποίηση δεδομένων, ενώ τα τυχαία δάση συνδυάζουν πολλά δέντρα απόφασης για να βελτιώσουν την ακρίβεια της πρόβλεψης.



Σχήμα 1.1: Δομή Δέντρου Απόφασης



Σχήμα 1.2: Δομή Τυχαίου Δάσους

Gradient Boosting: Εκπαιδεύει διαδοχικά υπομοντέλα, συνήθως δέντρα απόφασης, ώστε κάθε ένα να διορθώνει τα λάθη των προηγούμενων, βελτιώνοντας την ακρίβεια του συνολικού μοντέλου.

$$F_m(x) = F_{m-1}(x) + \rho_m h_m(x), \tag{1.5}$$

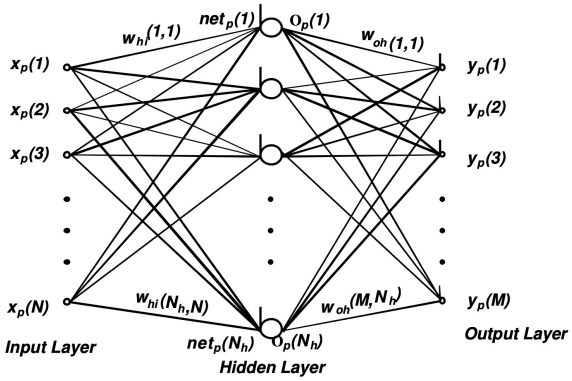
Gaussian Process: Το μοντέλο ουσιαστικά μοντελοποιεί τη βασική συνάρτηση ως μια Γκαουσιανή Διεργασία, που σημαίνει μια συλλογή τυχαίων μεταβλητών, οποιοδήποτε πεπερασμένο αριθμό των οποίων έχουν κοινή Γκαουσιανή κατανομή. Έτσι, το μοντέλο μπορεί όχι μόνο να προβλέψει την αναμενόμενη τιμή της συνάρτησης σε ένα δεδομένο σημείο, αλλά επίσης να εκτιμήσει την πιθανότητα του να είναι λάθος η πρόβλεψη αυτή. Μια στατιστική προσέγγιση που προέρχεται από τη βιβλιογραφία για τις χωρικές στατιστικές είναι το μοντέλο *GASP*. Το μοντέλο *GASP* θεωρεί την αποκριτική παραγωγή ως πολυδιάστατη κανονική κατανομή ή μια πραγματοποίηση μιας τυχαίας στοχαστικής διαδικασίας.

1.3.3 Νευρωνικά Δίκτυα

Εμπνευσμένα από τη βιολογία, τα νευρωνικά δίκτυα είναι ένα ισχυρό εργαλείο μηχανικής μάθησης που μπορεί να μάθει διάφορες εργασίες συσχέτισης και να προσεγγίσει σύνθετες μη γραμμικές συναρτήσεις.

Περσεπτρόνιο: Ο πιο βασικός τύπος νευρωνικού δικτύου, το οποίο εκτελεί δυαδική ταξινόμηση μέσω μιας σειράς δυαδικών εισόδων, πολλαπλασιασμών βαρών και συνόλων αποτελεσμάτων.

Πολυεπίπεδο Περσεπτρόνιο (*MLP*): Σύνθετο νευρωνικό δίκτυο με πολλαπλά επίπεδα νευρώνων που επιτρέπουν την επεξεργασία πιο σύνθετων μοτίβων.



Σχήμα 1.3: Δομή Πολυεπίπεδου Περσεπτρόνιου

Η εργασία αυτή εξετάζει λεπτομερώς αυτές τις τεχνικές και αναλύει πώς μπορούν να εφαρμοστούν για την πρόβλεψη των *KPI* και τη βελτιστοποίηση της ανάθεσης πόρων σε ασφαλιστικές εταιρείες. Η σύγκριση των αποδόσεων αυτών των μοντέλων αποκαλύπτει τα πλεονεκτήματα και τα μειονεκτήματα κάθε μεθόδου, καθοδηγώντας την μελλοντική εφαρμογή και έρευνα.

1.4 Προγραμματισμός Υπό Περιορισμούς και Ακέραιος Προγραμματισμός

1.4.1 Ορισμοί

Ο γραμμικός προγραμματισμός είναι μια μαθηματική τεχνική μοντελοποίησης στην οποία μια γραμμική συνάρτηση μεγιστοποιείται ή ελαχιστοποιείται υπό διάφορους πε-

ριορισμούς. Αυτή η μέθοδος έχει αποδειχθεί χρήσιμη στη βιομηχανική μηχανική, τον επιχειρησιακό προγραμματισμό και, σε μικρότερο βαθμό, στις κοινωνικές και φυσικές επιστήμες κατά τη λήψη ποσοτικών αποφάσεων.

Ένα ακέραιο πρόγραμμα είναι ένα γραμμικό πρόγραμμα όπου κάποιες ή όλες οι μεταβλητές αποφάσεων περιορίζονται να λάβουν μόνο ακέραιες τιμές. Αντίθετα, ο προγραμματισμός περιορισμών (*CP*) αναφέρεται στην ανεύρεση εφικτών λύσεων από ένα πολύ μεγάλο σύνολο υποψηφίων, όπου το πρόβλημα μπορεί να μοντελοποιηθεί με όρους αυθαίρετων περιορισμών.

Τα εργαλεία για τη διευκόλυνση της σχεδίασης και υλοποίησης προβλημάτων συνδυαστικής βελτιστοποίησης έχουν εξελιχθεί σημαντικά τα τελευταία 20 χρόνια. Ο στόχος τους είναι να μειώσουν σημαντικά τον χρόνο ανάπτυξης διατηρώντας την αποτελεσματικότητα των ειδικών προγραμμάτων.

1.4.2 Εργαλεία Βασισμένα σε Περιορισμούς

Η *OPL* είναι μια γλώσσα μοντελοποίησης που, όπως οι παραδοσιακές γλώσσες μοντελοποίησης, μοιράζεται υψηλού επιπέδου αλγεβρικές έννοιες. Περιέχει ειδικά χαρακτηριστικά για τη διαχείριση της σπανιότητας σε εφαρμογές μεγάλης κλίμακας, την υποστήριξη προβλημάτων χρονοπρογραμματισμού και κατανομής πόρων, και την ικανότητα ορισμού τακτικών και διαδικασιών αναζήτησης.

Επιπλέον, η *MiniZinc* είναι ένα εργαλείο για τη μοντελοποίηση προβλημάτων περιορισμών. Μπορεί να χειριστεί πιο σύνθετες εκφράσεις μέσω μιας διαδικασίας γνωστής ως αποσύνθεση.

1.4.3 Μοντελοποίηση Περιορισμών

Η μοντελοποίηση περιορισμών εγείρει δύο ζητήματα: την εκφραστικότητα και τη χρηστικότητα του εργαλείου καθώς και την εγγενή επεκτασιμότητά του. Η χρήση της γλώσσας για την απλοποίηση των προγραμμάτων και την αύξηση του προφίλ μοντελοποίησης σε κατάλληλο επίπεδο αφαίρεσης είναι δύσκολη.

Για παράδειγμα, το πρόβλημα της σειράς μαγείας περιλαμβάνει την εύρεση μιας σειράς αριθμών που πληρούν συγκεκριμένους περιορισμούς. Η μοντελοποίηση του προβλήματος με ένα εργαλείο προγραμματισμού περιορισμών απαιτεί την έκφραση των περιορισμών σε γραμμικούς συνδυασμούς στοιχειωδών περιορισμών.

1.4.4 Σχετικά Παραδείγματα

Οι τεχνικές προγραμματισμού περιορισμών εφαρμόζονται σε πλήθος προβλημάτων, ιδιαίτερα σε αυτά που αφορούν τον χρονοπρογραμματισμό και την κατανομή πόρων. Ένα από τα πιο χαρακτηριστικά παραδείγματα είναι ο προγραμματισμός πραγματικού χρόνου σε διασταυρώσεις. Σε αυτά τα προβλήματα, μικρές διακοπές στη ροή της κυκλοφορίας μπορούν να προκαλέσουν μεγάλες καθυστερήσεις, δημιουργώντας την ανάγκη για αποδοτικές λύσεις που μπορούν να διαχειριστούν τις συνεχείς αλλαγές και τις απαιτήσεις του συστήματος.

Ο προγραμματισμός υπό περιορισμούς χρησιμοποιείται επίσης ευρέως στην αλυσίδα εφοδιασμού για τη βελτιστοποίηση της διαχείρισης των αποθεμάτων και των δρομολογίων

διανομής. Τα προβλήματα αυτά περιλαμβάνουν την επιλογή των βέλτιστων δρομολογίων για τα οχήματα που διανέμουν προϊόντα, λαμβάνοντας υπόψη παραμέτρους όπως η χωρητικότητα των οχημάτων, οι χρόνοι παράδοσης και οι περιορισμοί του δικτύου διανομής.

1.5 Αντιμετώπιση του Προβλήματος

Σε αυτό το κεφάλαιο αναλύεται η πολυπλοκότητα τόσο του συστήματος όσο και του προβλήματος που αντιμετωπίζεται, ώστε ο αναγνώστης να αποκτήσει μια ολοκληρωμένη εικόνα. Το σύστημα χωρίζεται σε τέσσερα μέρη, καθένα από τα οποία είναι ζωτικής σημασίας για τη συνολική λύση.

1.5.1 Εννοιολογικός Ορισμός

Η παρούσα διπλωματική εργασία ασχολείται με τη βελτιστοποίηση τόσο του Προγράμματος Χρονοπρογραμματισμού (*Scheduling Solver*) όσο και του Μοντέλου Πρόβλεψης (*Surrogate Model*). Το πρόβλημα μπορεί να θεωρηθεί ότι ανήκει στις κατηγορίες κατανομής πόρων και χρονοπρογραμματισμού. Δεδομένου ότι το μοντέλο πρόβλεψης προβλέπει τα περιστατικά που εξυπηρετούνται και τον μέσο χρόνο εξυπηρέτησης για όλους τους δυνατούς συνδυασμούς των (M, P) σε διαφορετικές ώρες της ημέρας και υπό διαφορετικές συνθήκες προσομοίωσης, μπορεί να ειπωθεί ότι πρόκειται για ένα γενικευμένο πρόβλημα κατανομής πόρων. Στόχος είναι να εντοπιστεί ο βέλτιστος συνδυασμός των διαθέσιμων οχημάτων για κάθε ώρα, ώστε να επιτευχθούν οι στόχοι της εταιρείας.

1.5.2 Δεδομένα Εισόδου

Τα δεδομένα εισόδου για το πρόβλημα είναι τα ιστορικά δεδομένα των περιστατικών σε συγκεκριμένες ημέρες. Αυτά περιλαμβάνουν πληροφορίες όπως την ημερομηνία και ώρα του περιστατικού, τις συντεταγμένες, τον τύπο του οχήματος που απαιτείται, την εταιρεία που ανήκει το περιστατικό και τον χρόνο εξυπηρέτησης. Οι τύποι των διαθέσιμων οχημάτων είναι δύο: ¨Μοτοσικλέτες¨ και ¨Πλατφόρμες¨. Η κύρια διαφορά μεταξύ τους είναι ότι οι πλατφόρμες έχουν τη δυνατότητα ρυμούλκησης. Το σύστημα αποδέχεται συγκεκριμένους στόχους με τη μορφή Βασικών Δεικτών Απόδοσης (*KPIs*), όπως το ποσοστό των περιστατικών που εξυπηρετούνται σε λιγότερο από X λεπτά και ο μέσος χρόνος εξυπηρέτησης όλων των περιστατικών της εταιρείας.

Ο χρόνος εξυπηρέτησης ορίζεται ως ο χρόνος μεταξύ της κλήσης για το περιστατικό και της στιγμής άφιξης της οδικής βοήθειας. Για παράδειγμα, μια συγκεκριμένη εταιρεία μπορεί να έχει θέσει στόχους ώστε το $PercentageOverX = 80\%$ των περιστατικών της να εξυπηρετούνται σε χρόνο μικρότερο από $GoalMinutesX = 45'$, και αντίστοιχα το μέγιστο ποσοστό του $PercentageOverY = 3\%$ να εξυπηρετείται σε χρόνο μεγαλύτερο από $GoalMinutesY = 180'$. Αυτό επιβάλλει ένα πολύ συγκεκριμένο όριο στις μετρικές απόδοσης που πρέπει να έχει το επιλεγμένο μοντέλο, καθώς οι δυνατότητες του μοντέλου παίζουν ζωτικό ρόλο στο αν θα επιτευχθούν ή όχι οι στόχοι της εταιρείας.

PercentageUnderX	Percentage of accidents served in less than X minutes.
PercentageOverY	Percentage of accidents served in more than Y minutes.
AverageServiceTime	Average service time of all company incidents.
GoalMinutesX	Value of X
GoalMinutesY	Value of Y

Σχήμα 1.4

1.5.3 Προσομοιωτής

Ο προσομοιωτής μιμείται την διαχείριση των περιστατικών οδικής βοήθειας όπως αυτά συμβαίνουν σε πραγματικό χρόνο, με στόχο την παραγωγή δεδομένων για το μοντέλο πρόβλεψης. Στην πράξη, μπορεί να χρησιμοποιηθεί για να βρεθεί η συσχέτιση μεταξύ του αριθμού των πόρων και των επιτευχθέντων στόχων. Τα δεδομένα εισόδου είναι τα ιστορικά περιστατικά μιας χρονικής περιόδου και οι συνδυασμοί των διαθέσιμων αριθμών πλατφορμών και μοτοσικλετών. Ο προσομοιωτής παράγει το τριάδα (*M, P, KPI*) για κάθε χρονική περίοδο.

Ο *Driver Availability Generator* είναι ένα σύστημα που λαμβάνει ως είσοδο τις διάφορες καθορισμένες βάρδιες των οδηγών και υπολογίζει πόσοι και με τι είδους όχημα θα είναι διαθέσιμοι σε μια συγκεκριμένη χρονική περίοδο. Συστήματα όπως αυτό και ο προσομοιωτής θεωρούνται αυθαίρετα ότι λειτουργούν ιδανικά, και η πολυπλοκότητά τους είναι εκτός του πλαισίου της παρούσας διπλωματικής εργασίας. Ο ρόλος τους είναι σημαντικός, αλλά συμπληρωματικός στη συνολική λύση, η οποία εξαρτάται σε μεγάλο βαθμό από την απόδοση και την αποτελεσματικότητα του υποκατάστατου μοντέλου. Το τελευταίο, το οποίο αποτελεί το επίκεντρο των τριών πρώτων κεφαλαίων, εξηγείται παρακάτω.

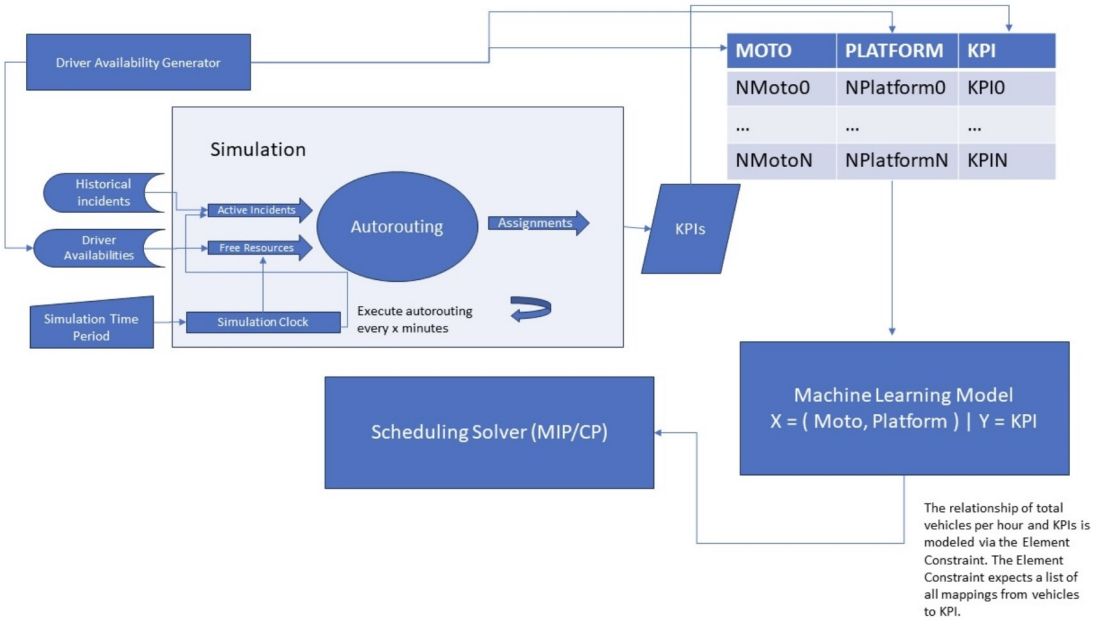
1.5.4 Μοντέλο Πρόβλεψης

Καθοριστικής σημασίας για τις ανάγκες του προβλήματος είναι η ακριβής εκτίμηση όσο το δυνατόν περισσότερων τριάδων (M, P, KPI), όπως εξηγείται στην υποενότητα 5.1.3. Υπάρχει ανάγκη για έρευνα, κατάλληλη επιλογή και εκπαίδευση μοντέλων μηχανικής μάθησης που θα παρέχουν σε σχετικά σύντομο χρονικό διάστημα ακριβή εκτίμηση των προαναφερθέντων KPI. Ουσιαστικά, λόγω των περιορισμών που προκύπτουν από τον χρόνο που απαιτείται για την εκτέλεση του προσομοιωτή, το μοντέλο τελικά θα πρέπει να αποδέχεται συγκεκριμένες τριάδες και να προβλέπει με σχετική ακρίβεια τα KPI για όλους τους υπόλοιπους συνδυασμούς. Οπτικά, υπάρχει μια συνάρτηση που έχει μια είσοδο που βελτιστοποιεί τις τιμές της στους άξονες των KPI ενώ ταυτόχρονα ελαχιστοποιεί τα χαμένα μέσα, δηλαδή τις πλατφόρμες και τις μηχανές. Στόχος του μοντέλου είναι, λαμβάνοντας ορισμένα σημεία της συνάρτησης (καμπύλης), να προβλέψει τα υπόλοιπα. Επιπλέον, το μοντέλο πρέπει να εκτελείται για συγκεκριμένα KPI κάθε φορά και για κάθε εταιρεία. Με άλλα λόγια, απαιτούνται διαφορετικές εκτελέσεις για το *PercentageUnderX* και για το *PercentageOverY* αντίστοιχα. Αξίζει να σημειωθεί ότι τέτοιες προβλέψεις χρειάζονται για κάθε ώρα κάθε ημερολογιακής ημέρας, προκειμένου να επιλέγονται αργότερα οι διαδρομές και οι οδηγοί για την συγκεκριμένη

χρονική περίοδο. Ο λόγος που οι προβλέψεις γίνονται ξεχωριστά για κάθε ώρα είναι για να απομονωθεί η σχέση μεταξύ των πόρων και των KPI χωρίς να επηρεάζεται από τη χωρική και χρονική κατανομή των περιστατικών.

1.5.5 Πρόγραμμα Χρονοπρογραμματισμού

Το Πρόγραμμα Χρονοπρογραμματισμού χρησιμοποιεί τις προβλέψεις του Υποκατάστατου Μοντέλου για να καταρτίσει ένα βέλτιστο πρόγραμμα αλλαγής των οδηγών. Ενσωματώνει διάφορους περιορισμούς, όπως οι βάρδιες των οδηγών και οι κανόνες της εταιρείας, για να εξασφαλίσει ότι οι στόχοι εξυπηρέτησης επιτυγχάνονται με τον καλύτερο δυνατό τρόπο.



Σχήμα 1.5: Δομή Συστήματος

1.5.6 Δοκιμή του Μοντέλου Πρόβλεψης

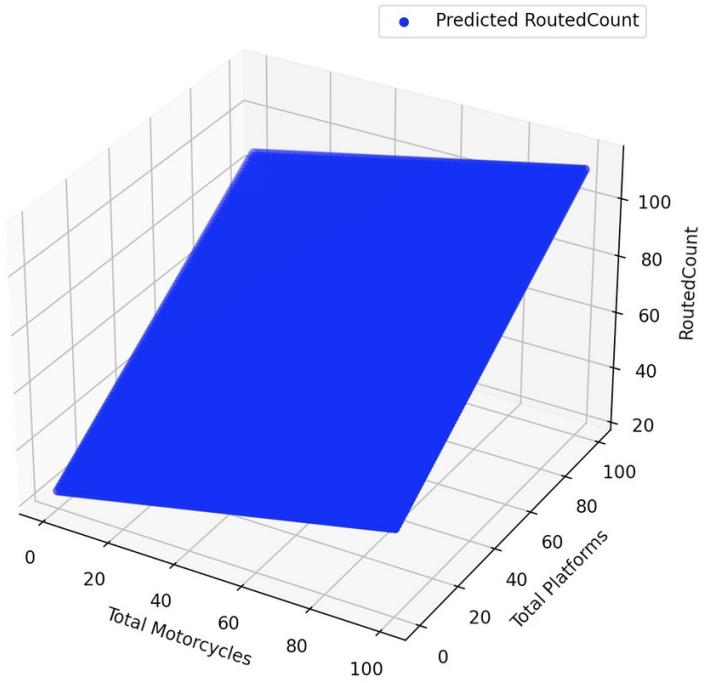
Για την κατάλληλη αντιμετώπιση του προβλήματος και την εύρεση του καλύτερου υποκατάστατου μοντέλου, δοκιμάστηκε και αξιολογήθηκε ένας αριθμός μοντέλων μηχανικής μάθησης με τα καθορισμένα πρότυπα απόδοσης που συζητήθηκαν στο κεφάλαιο 1. Παρόλο που μόνο ένα μικρό μέρος από αυτά αποδείχθηκε κατάλληλο για το πρόβλημά μας, για λόγους πληρότητας αναφέρονται τα περισσότερα. Κάθε μοντέλο εκπαιδεύτηκε στο 80% των δεδομένων και στη συνέχεια η προσαρμοστικότητά του επικυρώθηκε στο υπόλοιπο 20%. Όλη η εκπαίδευση των μοντέλων πραγματοποιήθηκε σε Πψτηον, αξιοποιώντας τη βιβλιοθήκη *scikit – learn* και τις δυνατότητές της.

Είναι άμεσα αντιληπτό ότι η φύση της λύσης του προβλήματος μας δεν είναι γραμμική. Εκτός από τις πολύπλοκες συσχετίσεις μεταξύ των παραμέτρων εισόδου και εξόδου, το μέγεθος του διανύσματος εισόδου X αποτελεί περιοριστικό παράγοντα για τα γραμμικά μοντέλα. Λογικά, μόνο μετά από πολλές απλοποιήσεις τα μοντέλα θα μπορούσαν να κάνουν αρκετά ακριβείς προβλέψεις ώστε να θεωρηθούν αποδεκτές, αλλά τότε η φύση του προβλήματος θα άλλαζε. Παρ' όλα αυτά, δοκιμάστηκαν οκτώ γραμμικά μοντέλα μηχανικής μάθησης και αξιολογήθηκε η ακρίβειά τους.

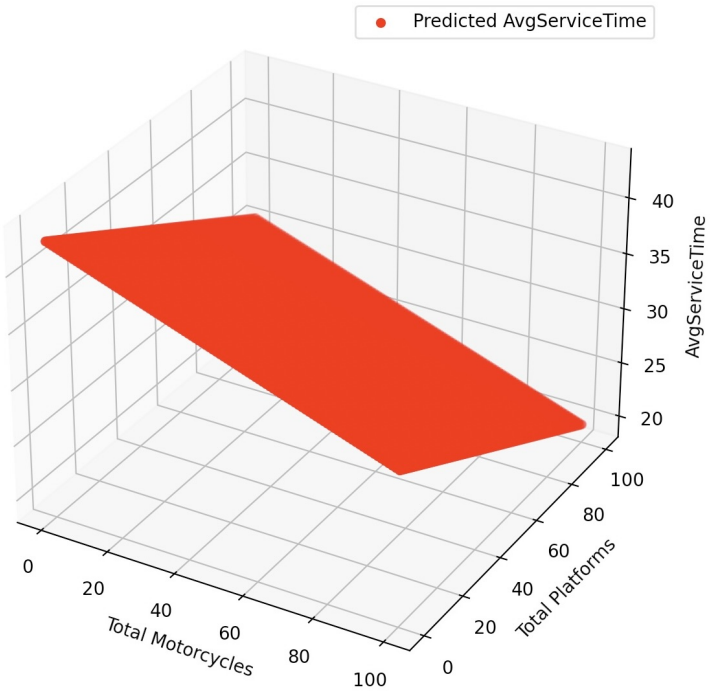
Το προφανές μοντέλο που δοκιμάστηκε πρώτο ήταν η κλασική γραμμική παλινδρόμηση. Ακολουθεί τη μαθηματική εξίσωση:

$$Y = Xb + \epsilon \tag{1.6}$$

όπου ϵ είναι ένας n-διάστατος πίνακας ανεξάρτητων και ταυτοσήμως κατανεμημένων τυχαίων μεταβλητών (συχνά θεωρείται Γκαουσιανή) με μέση τιμή 0 και διακύμανση σ^2, το X είναι ένας np πίνακας με μοναδικούς συσχετισμένους διανύσματα ως τις γραμμές του, και το Y είναι ένας n-διάστατος πίνακας ανεξάρτητων απαντήσεων. Στο παράδειγμά μας, το Y παίρνει δύο ξεχωριστές τιμές, τον αριθμό των περιστατικών που εξυπηρετούνται (*RoutedCount*) και τον μέσο χρόνο εξυπηρέτησης κάθε περιστατικού (*AvgServiceTime*). Το μοντέλο εκπαιδεύεται δύο φορές, μία για κάθε μεταβλητή. Χρησιμοποιεί τη μέθοδο των ελαχίστων τετραγώνων (*OLS*) που στοχεύει ουσιαστικά στην ελαχιστοποίηση του Υπολοίπου Άθροισμα των Τετραγώνων (*RSS*). Μετά την εκπαίδευση, οι προβλέψεις ήταν γραφικά οι εξής:



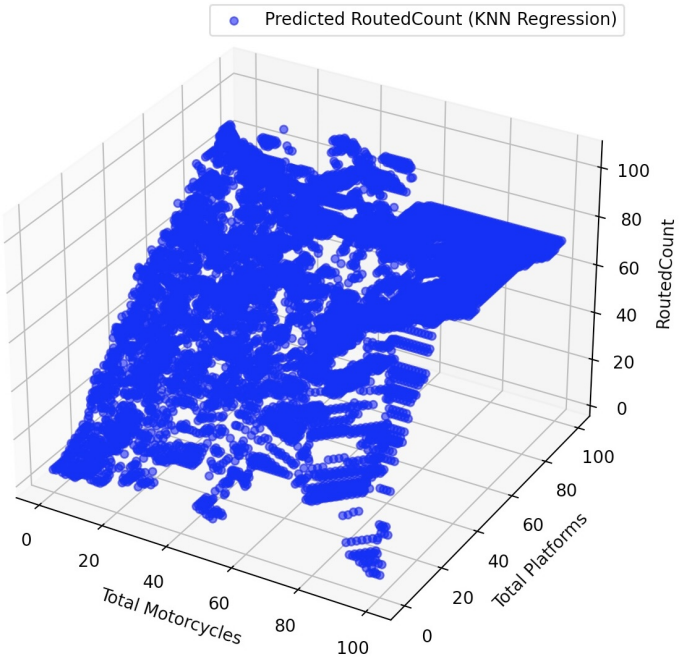
Σχήμα 1.6: Πρόβλεψη Γραμμικής Παλινδρόμησης για περιστατικά που εξυπηρετούνται



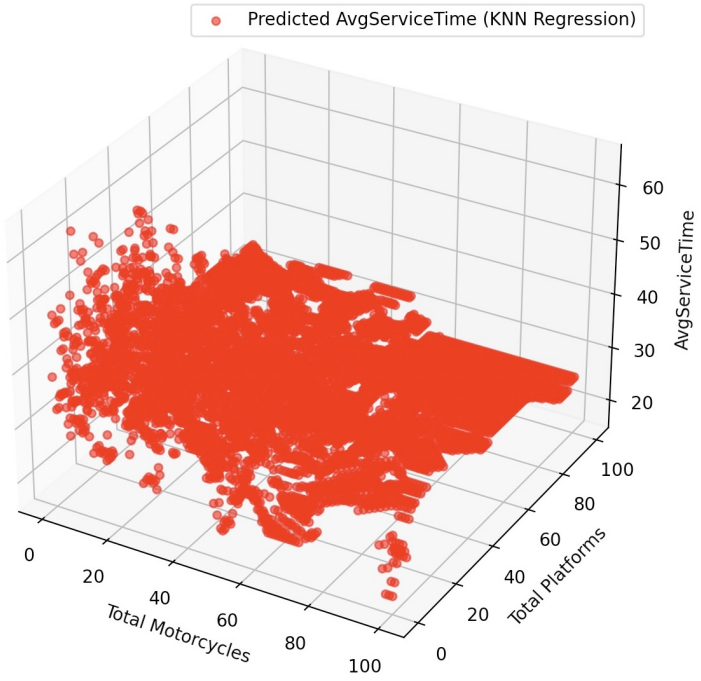
Σχήμα 1.7: Πρόβλεψη Γραμμικής Παλινδρόμησης για τον μέσο χρόνο εξυπηρέτησης περιστατικών.

Όπως μπορεί να δει κανείς, η πρόβλεψη είναι αρκετά λογική, με τον αριθμό των περιστατικών που εξυπηρετούνται να ελαχιστοποιείται (=0) όταν (M, P) = (0, 0) και να ακολουθεί μια γραμμική αυξανόμενη πορεία μέχρι το μέγιστο του στο (M, P) = (100, 100). Μια παρόμοια δομή ακολουθείται και για την πρόβλεψη του μέσου χρόνου εξυπηρέτησης, όπου ελαχιστοποιείται για τον μέγιστο αριθμό διαθέσιμων πόρων και μεγιστοποιείται για τον ελάχιστο.

Δοκιμάστηκαν διάφορα μη γραμμικά μοντέλα όπως η πολυωνυμική παλινδρόμηση, το μοντέλο *K − NearestNeighbors* (KNN) και ο *Decision Tree Regressor*. Το μοντέλο *K−Nearest Neighbors* αποδείχθηκε πιο προσαρμοστικό, ενώ ο *Decision Tree Regressor* παρουσίασε βελτιωμένες προβλέψεις.



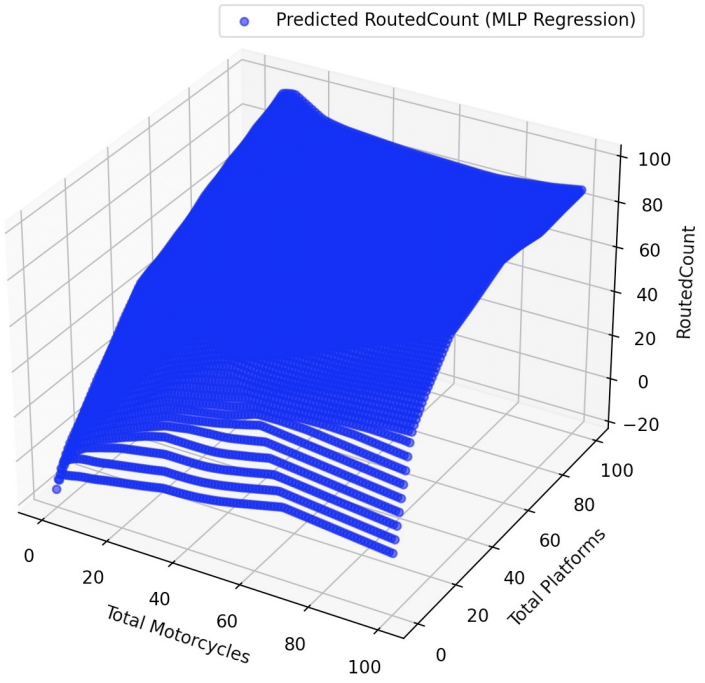
Σχήμα 1.8: Πρόβλεψη του μοντέλου KNN με K = 5 για τον αριθμό των περιστατικών που εξυπηρετούνται.



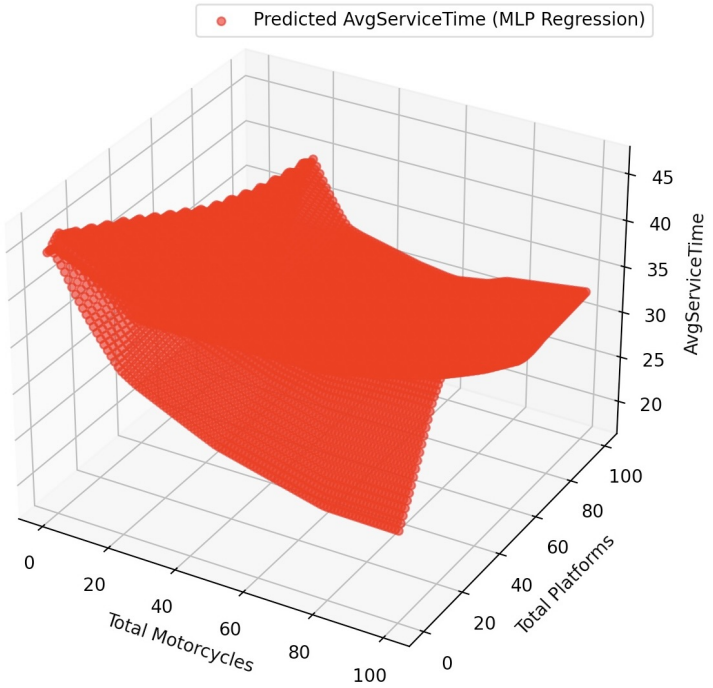
Σχήμα 1.9: Πρόβλεψη του μοντέλου KNN με K = 5 για τον μέσο χρόνο εξυπηρέτησης των περιστατικών.

Η περίπτωση του πολυστρωματικού αντιληπτικού (*MLP*) είναι αρκετά ενδιαφέρουσα. Αν και συνήθως τα νευρωνικά δίκτυα χρησιμοποιούνται για την επίλυση πιο σύνθετων

προβλημάτων, η απόδοση του *MLP* ήταν χαμηλότερη σε σύγκριση με άλλα μη γραμμικά μοντέλα.



Σχήμα 1.10: Πρόβλεψη του *MLP* για τον αριθμό των περιστατικών που εξυπηρετούνται.

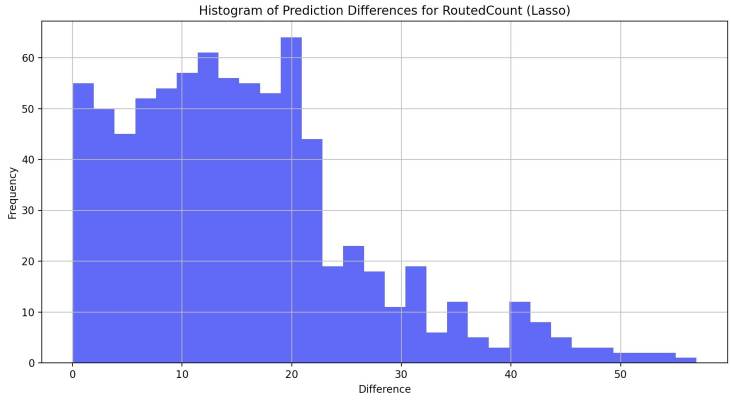


Σχήμα 1.11: Πρόβλεψη του *MLP* για τον μέσο χρόνο εξυπηρέτησης των περιστατικών.

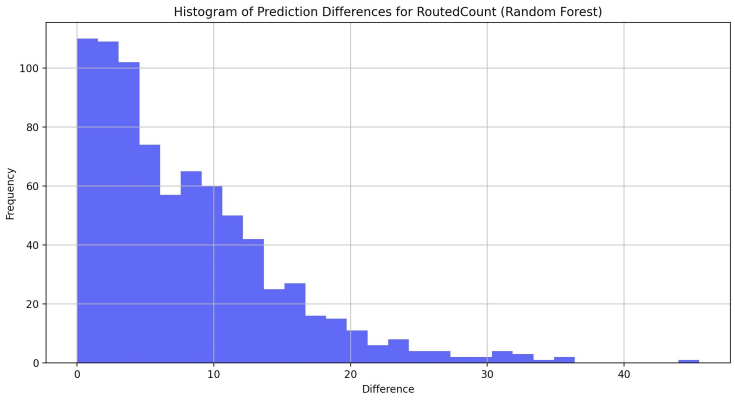
1.5.7 Ακραίες Περιπτώσεις

Όπως είναι προφανές από την προηγούμενη ενότητα, η εργασία μας έχει παράγει μια αξιόπιστη προσέγγιση της ιδανικής πρόβλεψης έχοντας βελτιώσει το προηγούμενο όριο απόδοσης. Το ερώτημα τώρα γίνεται αν η ακρίβεια μπορεί να βελτιωθεί περαιτέρω. Στην προϋπάρχουσα λύση του προβλήματος, υπήρχε σημαντικός αριθμός ακραίων περιπτώσεων που ζημίωναν τη συνολική πρόβλεψη του αλγορίθμου. Το επόμενο βήμα είναι η αναγνώριση αυτών των ακραίων περιπτώσεων και η διερεύνηση τρόπων μείωσής τους.

Αρχικά, για κάθε εκπαιδευμένο μοντέλο δημιουργήθηκε ένα ιστόγραμμα που δείχνει τις διαφορές μεταξύ των προβλέψεων και των πραγματικών δεδομένων από το σετ δοκιμών. Στόχος είναι η ελαχιστοποίηση του αριθμού των περιπτώσεων μεγαλύτερων από μια συγκεκριμένη τιμή, ανάλογα με τη μεταβλητή εξόδου που προβλέπεται.



Σχήμα 1.12: Ιστόγραμμα διαφορών για το *Lasso* μοντέλο.



Σχήμα 1.13: Ιστόγραμμα διαφορών για το *Random Forest* μοντέλο.

Το ζήτημα αυτό μετριάστηκε όταν περάσαμε στο πιο αποτελεσματικό μοντέλο, το Ρανδομ Φορεστ. Δεδομένου του ανώτερου R^2 *score*, αυτό δείχνει μια πιο στενή προσαρμογή του μοντέλου στα δεδομένα, μειώνοντας έτσι την ανεξήγητη διακύμανση γύρω από τη μέση τιμή της μεταβλητής απόκρισης.

Οι διαφορές έχουν μειωθεί σημαντικά αλλά εξακολουθούν να καταλαμβάνουν ανεπιθύμητες τιμές. Ιδανικά, θα θέλαμε οι περισσότερες προβλέψεις να έχουν διαφορά με

τα πραγματικά δεδομένα μικρότερη από 10, με ορισμένες να βρίσκονται στο διάστημα μεταξύ 10 και 15. Αυτό θα ήταν ένα ιδιαίτερα στενό περιθώριο που θα ελαχιστοποιούσε τα λάθη στις βάρδιες των οδηγών.

1.6 Βελτιστοποίηση Μοντέλου: Παράμετροι και Ακραίες Περιπτώσεις

1.6.1 Μια Πιο Ολιστική Προσέγγιση

Ας διευκρινίσουμε ότι υπάρχουν δύο βασικές φιλοσοφίες για την προσέγγιση της λύσης. Η πρώτη αφορά την εκπαίδευση των μοντέλων μας βασισμένη σε (μόνο) δύο παραμέτρους, το ζεύγος (M, P). Αυτή είναι η προσέγγιση που ακολουθήσαμε στο Κεφάλαιο 5 και έχει τα πλεονεκτήματα της ταχύτητας και της ακρίβειας σε όρους χρονολογικής σειράς. Μπορούμε να εκτελέσουμε το μοντέλο περισσότερες φορές για να λάβουμε τα καθαρότερα δυνατά αποτελέσματα. Ωστόσο, είναι προφανές ότι αυτή η μέθοδος, ακόμη και όταν δοκιμάζονται πολλά μοντέλα, δεν μπορεί να παρέχει την καλύτερη δυνατή λύση. Αν θέλουμε να επιτύχουμε έναν καλύτερο συνδυασμό, τότε πρέπει να επανεξετάσουμε τους βασικούς πυλώνες του πρώτου σχεδίου μας.

Η δεύτερη προσέγγιση προχωρά προς μια πιο ολιστική ανασκόπηση που λαμβάνει υπόψη διάφορους άλλους καθοριστικούς παράγοντες, όπως την ώρα της ημέρας, την κυκλοφορία, τα συνολικά περιστατικά που συμβαίνουν σε αυτήν την συγκεκριμένη ώρα, καθώς και έναν νέο δείκτη βαρύτητας που θα συζητήσουμε αργότερα. Φυσικά, δεν μπορούσαμε να διατηρήσουμε κάθε παράμετρο στην τελική εκπαίδευση του μοντέλου, καθώς αυτό θα το καθιστούσε ευάλωτο στην υπερπροσαρμογή. Για παράδειγμα, η ώρα της ημέρας μπορεί να περιλαμβάνει έμμεσα την κυκλοφορία, οπότε ακολουθώντας απλή λογική και τεχνικές δοκιμής και σφάλματος καταλήξαμε στην τελική λύση. Αν και αυτή μπορεί να είναι μια πιο χρονοβόρα/πόρων απαιτητική μέθοδος, δεν υπερβαίνει τα απαγορευτικά όρια. Με άλλα λόγια, αυτό είναι ένα τίμημα που μπορούμε να αντέξουμε για πολύ βελτιωμένη ποιότητα πρόβλεψης.

Τα διαθέσιμα δεδομένα από τον προσομοιωτή (5.1) ήταν υπερβολικά για τα μοντέλα μας. Η λύση περιλάμβανε την απομόνωση των πιο σημαντικών πληροφοριών και τη μετατροπή τους σε ένα μεγάλο αρχείο *.json* που θα μπορούσε να χρησιμοποιηθεί ως εκπαιδευτικό σύνολο δεδομένων.


```
{
    "TotalM": 5,
    "TotalP": 6,
    "RoutedCount": 12,
    "TotalV": 11,
    "AvgServiceTime": 30.704166666666666,
    "NonEmptyLines": 43,
    "hour": 6,
    "lastFreeV": 7,
    "HeavinessMetric": 0.36363636363636365
},
```


Σχήμα 1.14: *JSON object* στο *dataset*

1.6.2 Ανάλυση Δεδομένων

Στην όλη διαδικασία, χρησιμοποιούσαμε δεδομένα σε μορφή *.json*, καθώς έχει αποδειχθεί ότι είναι η πιο αποτελεσματική και αξιόπιστη μέθοδος. Ακολουθήθηκαν δύο γενικές μέθοδοι. Η πρώτη υπήρχε ουσιαστικά στη λύση του συστήματος, αλλά αποδείχθηκε απαραίτητο να δημιουργηθεί μια δεύτερη, πιο ολιστική. Αυτές οι δύο απαιτούσαν δύο διαφορετικά εκπαιδευτικά σύνολα δεδομένων, καθένα με τις δικές του ιδιαιτερότητες. Γενικά, στη μηχανική μάθηση τα μοντέλα, εκτός από τις δικές τους αλγοριθμικές δομές, εξαρτώνται πραγματικά από τα αντίστοιχα εκπαιδευτικά δεδομένα που τους δίνονται. Έτσι, σε πολλές περιπτώσεις, η διαφορά μεταξύ των επιδόσεων διαφόρων μοντέλων οφείλεται κυρίως σε ζητήματα πληρότητας του συνόλου δεδομένων.

Πρέπει να σημειωθεί ότι σε αυτή την ενότητα το μοντέλο μηχανικής μάθησης που εκπαιδεύεται είναι ο *Random Forest Regressor*. Βασισμένο στα προηγούμενα κεφάλαια, την αυξημένη απόδοσή του και την αποτελεσματικότητα του χρόνου εκτέλεσης, αποδείχθηκε ότι είναι η πιο κατάλληλη επιλογή για το πρόβλημά μας.

Στην πρώτη περίπτωση, όπως εξηγήθηκε παραπάνω, ακολουθήσαμε τη λογική ότι ο αριθμός των περιστατικών και ο μέσος χρόνος εξυπηρέτησης εξαρτώνται κυρίως από το ζεύγος (M, P) και τις διάφορες τιμές του. Αυτή η λογική απαιτεί ένα αρχείο *.json* με πολλά αντικείμενα, καθένα με διαφορετικά αποτελέσματα προσομοίωσης. Αν θέλαμε να προβλέψουμε το ¨*RoutedCount*¨ για μια συγκεκριμένη ώρα της ημέρας, τροφοδοτούσαμε το μοντέλο με πολλές τιμές ¨*RoutedCount*¨, καθεμία από τις οποίες αντιστοιχούσε σε διαφορετικό συνδυασμό (M, P). Στη συνέχεια, με βάση φυσικά τον αντίστοιχο αλγόριθμο, το μοντέλο μπορούσε να βρει πρότυπα στα δεδομένα και να προβλέψει την τιμή για τους υπόλοιπους συνδυασμούς.

Ανάλογα με την τιμή που θέλαμε να προβλέψουμε μεταξύ ¨*RoutedCount*¨ και ¨*AvgServiceTime*¨, βεβαιωνόμασταν ότι το μοντέλο έπαιρνε υπόψη το σωστό για να κάνει όσο το δυνατόν πιο ακριβή πρόβλεψη. Το επόμενο βήμα ήταν η αξιολόγηση των προαναφερθέντων προβλέψεων. Όπως εξηγήθηκε, οι κύριες μετρικές απόδοσης που χρησιμοποιήθηκαν ήταν το R^2 *score* και το *MSE*. Ενώ αξιόπιστες, μπορούν να επηρεαστούν υπερβολικά από την ποιότητα των δεδομένων και ως εκ τούτου δεν μπορούσαν να είναι η μόνη πηγή μιας μετρικής. Ήταν προφανές ότι έπρεπε να δημιουργήσουμε τις δικές μας μετρικές, ειδικά αν χρειαζόταν να εντοπίσουμε συγκεκριμένα προβλήματα, όπως διάφορες ακραίες περιπτώσεις, και να τα αντιμετωπίσουμε αναλόγως.

Έτσι, χρησιμοποιώντας τη διασταυρούμενη επικύρωση, σχεδιάσαμε ένα σενάριο *Python* που χρησιμοποίησε το 80% του συνόλου δεδομένων για την εκπαίδευση του μοντέλου και το υπόλοιπο 20% για τη δοκιμή των απαντήσεών του. Συγκρίνοντας τις προβλέψεις με τις αντίστοιχες τιμές που δημιουργήθηκαν από τον προσομοιωτή, μπορέσαμε να εντοπίσουμε τις διάφορες ακραίες περιπτώσεις και να βρούμε τους λόγους που τις δημιούργησαν. Το προϊόν της παραπάνω διαδικασίας ήταν ένα αρχείο *.json* με αντικείμενα που περιείχαν πληροφορίες σχετικά με τη διαφορά των προαναφερθέντων ζευγών τιμών. Στη συνέχεια, εμπειρικά και χρησιμοποιώντας πάλι τις τυπικές μετρικές απόδοσης του R^2 *score* και του *MSE*, εντοπίσαμε την ακριβή διαφορά πάνω από την οποία η πρόβλεψη θεωρήθηκε ακραία περίπτωση. Για το *RoutedCount* αποδείχθηκε ότι είναι 10, ενώ για το *AvgServiceTime* 15.

1.6.3 Βελτιστοποίηση Παραμέτρων

Η βελτιστοποίηση παραμέτρων είναι μια κρίσιμη διαδικασία για τη βελτίωση της απόδοσης των μοντέλων μηχανικής μάθησης. Στην ενότητα αυτή, περιγράφεται η διαδικασία που ακολουθήθηκε για την εύρεση των βέλτιστων παραμέτρων του υποκατάστατου μοντέλου, με σκοπό την αύξηση της ακρίβειας και της αποτελεσματικότητάς του.

Η διαδικασία της βελτιστοποίησης περιλαμβάνει διάφορα βήματα, καθένα από τα οποία συμβάλλει στην τελική απόδοση του μοντέλου. Αυτά τα βήματα περιλαμβάνουν την αρχική ρύθμιση των παραμέτρων, τη δοκιμή διαφορετικών συνδυασμών και την αξιολόγηση της απόδοσης με τη χρήση μετρικών όπως το R^2 και το *MSE*.

Ξεκινήσαμε με τις προκαθορισμένες τιμές των παραμέτρων για το μοντέλο *Random Forest Regressor*. Αυτές οι τιμές περιλάμβαναν τον αριθμό των δέντρων στο δάσος, το μέγιστο βάθος των δέντρων, και τον ελάχιστο αριθμό δειγμάτων που απαιτούνται για τη διαίρεση ενός κόμβου.

Χρησιμοποιώντας μια διαδικασία πλέγματος αναζήτησης (*grid search*), δοκιμάσαμε διάφορους συνδυασμούς παραμέτρων. Αυτή η μέθοδος επιτρέπει τη συστηματική δοκιμή κάθε πιθανού συνδυασμού, προκειμένου να βρεθούν οι παράμετροι που προσφέρουν την καλύτερη απόδοση.

Για κάθε συνδυασμό παραμέτρων, αξιολογήσαμε την απόδοση του μοντέλου χρησιμοποιώντας το σετ επικύρωσης. Οι κύριες μετρικές που χρησιμοποιήθηκαν ήταν το R^2 και το *MSE*. Οι τιμές αυτές μας επέτρεψαν να κατανοήσουμε πόσο καλά το μοντέλο προσαρμόζεται στα δεδομένα και πόσο ακριβείς είναι οι προβλέψεις του.

Η αρχική ρύθμιση των παραμέτρων περιλάμβανε: *nestimators*: Ξεκινήσαμε με 100 δέντρα και αυξήσαμε τον αριθμό μέχρι να μην παρατηρηθεί σημαντική βελτίωση στην απόδοση. *maxdepth*: Ξεκινήσαμε με μικρό βάθος και σταδιακά το αυξήσαμε για να αποφύγουμε την υπερπροσαρμογή. *minsamplesplit*: Ξεκινήσαμε με την ελάχιστη τιμή 2 και δοκιμάσαμε μεγαλύτερες τιμές για να βελτιώσουμε τη γενίκευση του μοντέλου.

Για την πλέγμα αναζήτησης, δημιουργήσαμε ένα πλέγμα τιμών για κάθε παράμετρο και εκτελέσαμε το μοντέλο για κάθε συνδυασμό. Αυτή η διαδικασία μας επέτρεψε να εντοπίσουμε τον συνδυασμό που προσέφερε την καλύτερη απόδοση.

Οι τελικές τιμές των παραμέτρων που προέκυψαν από τη διαδικασία βελτιστοποίησης ήταν:

nestimators: 150, *maxdepth*: 10, *minsamplesplit*: 5.

Με αυτές τις παραμέτρους, το μοντέλο *Random Forest Regressor* παρουσίασε τη βέλτιστη απόδοση, με βελτιωμένα R^2 και *MSE* σε σύγκριση με τις αρχικές τιμές.

Η διαδικασία βελτιστοποίησης παραμέτρων είναι μια σημαντική πτυχή της ανάπτυξης μοντέλων μηχανικής μάθησης, καθώς διασφαλίζει ότι το μοντέλο είναι όσο το δυνατόν πιο αποδοτικό και ακριβές. Στην παρούσα εργασία, η βελτιστοποίηση των παραμέτρων του *Random Forest Regressor* είχε ως αποτέλεσμα ένα πιο ισχυρό και αξιόπιστο μοντέλο για την πρόβλεψη των KPI.

1.7 Η Περίπτωση της *OPL*: Απλοποίηση Σύνθετων Εργασιών Δρομολόγησης Οχημάτων και Διαχείρισης Χωρητικότητας

Σε όλη τη διάρκεια αυτής της διατριβής, η έμφαση έχει δοθεί στη βελτιστοποίηση της ποιότητας των προβλέψεων του μοντέλου μηχανικής μάθησης, που παρέχουν τη βάση για τον προγραμματισμό του επιλυτή. Όπως περιγράφηκε στο κεφάλαιο 4, το τελευταίο μέρος του αλγορίθμου αποτελείται από ένα ακέραιο πρόγραμμα που αντιμετωπίζει το πρόβλημα δρομολόγησης οχημάτων. Ουσιαστικά, πρόκειται για ένα σύνθετο πρόβλημα προγραμματισμού περιορισμών και η λύση είναι γραμμένη σε *Python*, αξιοποιώντας αποτελεσματικά τις λειτουργικότητες των *OR − Tools*. Σε αυτό το κεφάλαιο θα επικεντρωθούμε περισσότερο σε αυτό το τελευταίο τμήμα και πώς μπορεί να βελτιστοποιηθεί. Θα αποδειχθεί ότι η Γλώσσα Προγραμματισμού Βελτιστοποίησης (*OPL*) είναι ένα ανώτερο εργαλείο για την επίλυση σύνθετων προβλημάτων δρομολόγησης οχημάτων και διαχείρισης χωρητικότητας, σε σύγκριση με το ευρέως χρησιμοποιούμενο πλαίσιο *OR − Tools*. Μέσα από μια λεπτομερή μελέτη περίπτωσης, αυτό το κεφάλαιο δείχνει πώς το *OPL* όχι μόνο βελτιώνει την ευκρίνεια και τη συντηρησιμότητα του μοντέλου, αλλά επίσης βελτιστοποιεί την απόδοση στην επίλυση προβλημάτων μεγάλης κλίμακας.

1.7.1 Ο Ρόλος των *OR − Tools* στην Τρέχουσα Λύση Δρομολόγησης Οχημάτων

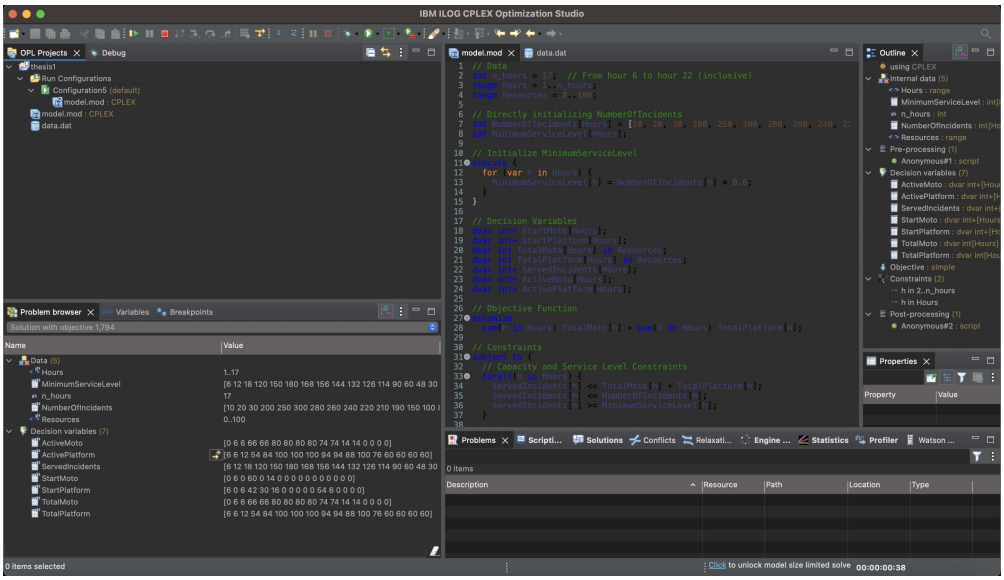
Στην επιχειρησιακή έρευνα και βελτιστοποίηση, η μοντελοποίηση παίζει κρίσιμο ρόλο στη διαμόρφωση πραγματικών προβλημάτων σε μαθηματικές αναπαραστάσεις που μπορούν να λυθούν υπολογιστικά. Τα *OR − Tools*, μια σουίτα λογισμικού ανοιχτού κώδικα που αναπτύχθηκε από την *Google*, παρέχουν μια ολοκληρωμένη πλατφόρμα για τη μοντελοποίηση και επίλυση συνδυαστικών προβλημάτων βελτιστοποίησης. Υποστηρίζει ένα ευρύ φάσμα τύπων προβλημάτων, όπως γραμμικός προγραμματισμός (*LP*), μεικτός ακέραιος προγραμματισμός (*MIP*), προγραμματισμός περιορισμών (*CP*) και προβλήματα δρομολόγησης οχημάτων (*VRP*). Αξιοποιώντας ισχυρούς επιλύτες και ένα ευέλικτο *API*, τα *OR − Tools* επιτρέπουν στους ερευνητές και επαγγελματίες να ορίζουν περιορισμούς, στόχους και μεταβλητές αποφάσεων με ευκολία, διευκολύνοντας την ανάπτυξη αποδοτικών και αποτελεσματικών μοντέλων βελτιστοποίησης.

Ένα τυπικό διαδικασία μοντελοποίησης στα *OR − Tools* ξεκινά με τον ορισμό των μεταβλητών αποφάσεων, που αντιπροσωπεύουν τις επιλογές που πρέπει να γίνουν. Αυτές οι μεταβλητές υπόκεινται σε περιορισμούς, αντικατοπτρίζοντας τους περιορισμούς ή απαιτήσεις του προβλήματος, και μια αντικειμενική συνάρτηση που ποσοτικοποιεί τον στόχο, όπως η ελαχιστοποίηση του κόστους ή η μεγιστοποίηση του κέρδους. Τα *OR − Tools* παρέχουν ένα διαισθητικό περιβάλλον για την εξειδίκευση αυτών των στοιχείων χρησιμοποιώντας διάφορες γλώσσες προγραμματισμού, όπως *Python*, *C++* και *Java*. Μόλις το μοντέλο οριστεί, τα *OR − Tools* χρησιμοποιούν *state − of − the − art* αλγόριθμους και επιλύτες για να βρουν βέλτιστες ή σχεδόν βέλτιστες λύσεις, προσφέροντας ανθεκτική απόδοση και επεκτασιμότητα. Η αρθρωτή αρχιτεκτονική των *OR − Tools* επιτρέπει επίσης την ενσωμάτωση προσαρμοσμένων ευρετικών και τεχνικών βελτιστοποίησης, καθιστώντας τα ένα ευέλικτο εργαλείο για την αντιμετώπιση σύνθετων προκλήσεων βελτιστοποίησης σε διάφορους τομείς.

1.7.2 Μετάφραση στην Γλώσσα Προγραμματισμού Βελτιστοποίησης (*OPL*)

Η εγκατάσταση του Ολοκληρωμένου Περιβάλλοντος Ανάπτυξης για το *OPL* και η μετάφραση ενός μοντέλου από την *Python* στο *OPL* απαιτεί μια δομημένη προσέγγιση, που ξεκινά με την εγκατάσταση και διαμόρφωση του απαραίτητου λογισμικού. Αρχικά, πρέπει να αποκτηθεί το *IBM ILOG CPLEX Optimization Studio*, που περιλαμβάνει το *OPL IDE*. Αυτό το λογισμικό μπορεί να ληφθεί από τον επίσημο ιστότοπο της *IBM* και πρέπει να αποκτηθεί μια κατάλληλη άδεια για να ξεκλειδωθούν όλες οι δυνατότητές του. Μόλις εγκατασταθεί το λογισμικό, το επόμενο βήμα είναι να διαμορφώσετε το *IDE*. Αυτό περιλαμβάνει τη ρύθμιση του περιβάλλοντος ώστε να αναγνωρίζει τα αρχεία και τις εξαρτήσεις του έργου, που είναι κρίσιμα για την αποτελεσματική ανάπτυξη και αποσφαλμάτωση.

Η αρχική ρύθμιση στο *OPL IDE* περιλαμβάνει τη δημιουργία ενός νέου έργου. Αυτή η διαδικασία ξεκινά με την επιλογή 'Νέο Έργο' και την εξειδίκευση του τύπου του έργου, συνήθως ένα *'OPL Project'*. Μέσα στο νεοδημιουργημένο έργο, θα πρέπει να οργανωθούν κατάλληλα τα αρχεία μοντέλου και δεδομένων. Για παράδειγμα, το κύριο αρχείο μοντέλου, που συχνά ονομάζεται *model.mod*, και το αρχείο δεδομένων, *data.dat*, πρέπει να τοποθετηθούν στον κατάλογο του έργου. Η διασφάλιση ότι το έργο είναι σωστά ρυθμισμένο μέσα στο *IDE* επιτρέπει την απρόσκοπτη εκτέλεση και δοκιμή του μοντέλου. Επιπλέον, η ρύθμιση του έργου ως ενεργού έργου και η σωστή διαμόρφωση των διαμορφώσεων εκτέλεσης είναι απαραίτητες για μια επιτυχημένη εγκατάσταση.



Σχήμα 1.15: *CPLEX studio*

Η μετάφραση του μοντέλου από την *Python* στο *OPL* απαιτεί προσεκτική εξέταση των συντακτικών και λειτουργικών διαφορών μεταξύ των δύο γλωσσών. Το πρώτο βήμα σε αυτή τη διαδικασία μετάφρασης είναι να κατανοήσετε τη δομή και τους περιορισμούς του μοντέλου *Python*. Βασικά στοιχεία όπως η αρχικοποίηση δεδομένων, οι μεταβλητές αποφάσεων, οι περιορισμοί και η αντικειμενική συνάρτηση πρέπει να αναγνωριστούν και να αντιστοιχιστούν στα αντίστοιχα του *OPL*. Στο *OPL*, τα δεδομένα συχνά αρχικοποιούνται μέσα στο αρχείο μοντέλου ή σε ένα ξεχωριστό αρχείο δεδομένων, και αυτά

τα δεδομένα χρησιμοποιούνται στη συνέχεια για τον ορισμό μεταβλητών αποφάσεων και περιορισμών. Η αντικειμενική συνάρτηση στο *OPL* καθορίζεται χρησιμοποιώντας τις λέξεις-κλειδιά μινιμιζε ή μαξιμιζε, παρόμοια με τη διατύπωση στην *Python* αλλά τηρώντας τη σύνταξη του *OPL*.

Κατά τη μετάφραση, είναι επίσης σημαντικό να ενσωματωθούν εκτελεστικά μπλοκ για την ανάλυση μετά τη λύση, παρόμοια με τη χρήση δηλώσεων πριντ στην *Python*. Το εκτελεστικό μπλοκ στο *OPL* επιτρέπει ενέργειες να εκτελούνται μετά την εύρεση λύσης από τον επιλύτη, όπως η εκτύπωση αποτελεσμάτων ή η εκτέλεση πρόσθετων υπολογισμών. Μετά τη μετάφραση του μοντέλου *Python* στη σύνταξη του *OPL*, το μοντέλο πρέπει να δοκιμαστεί διεξοδικά μέσα στο *OPL* ΙΔΕ για να διασφαλιστεί ότι συμπεριφέρεται όπως αναμένεται. Αυτό περιλαμβάνει την εκτέλεση του μοντέλου με διάφορα σύνολα δεδομένων, τον έλεγχο για σφάλματα και την επαλήθευση ότι οι περιορισμοί και η αντικειμενική συνάρτηση είναι σωστά υλοποιημένα. Ακολουθώντας συστηματικά αυτά τα βήματα, μπορεί κανείς να μεταφράσει επιτυχώς ένα μοντέλο *Python* στο *OPL*, αξιοποιώντας τα ισχυρά χαρακτηριστικά των εργαλείων βελτιστοποίησης της *IBM* για ενισχυμένες δυνατότητες επίλυσης προβλημάτων.

1.7.3 Πειραματικά Αποτελέσματα

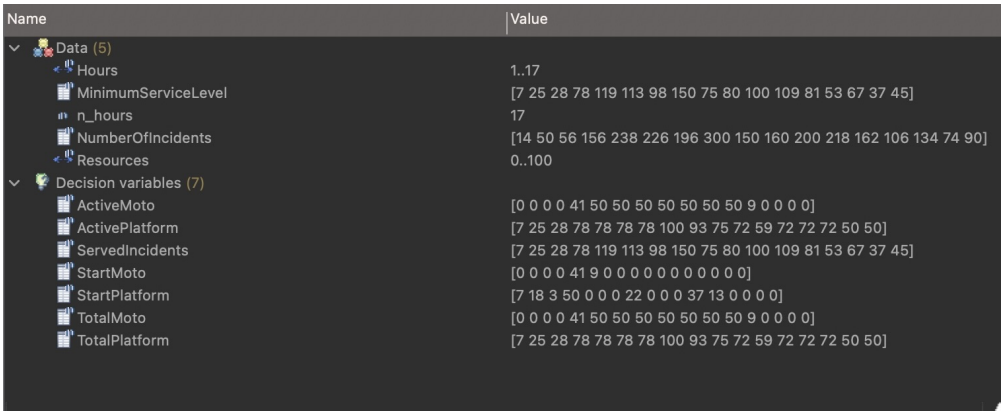
Τα παραγόμενα παραδείγματα για το πρόβλημα δρομολόγησης οχημάτων σχεδιάστηκαν για να προσομοιώνουν ρεαλιστικά μοτίβα κυκλοφορίας και περιστατικά καθ' όλη τη διάρκεια μιας τυπικής ημέρας. Ο αριθμός των περιστατικών διαφοροποιήθηκε ανάλογα με την ώρα της ημέρας για να αντικατοπτρίσει τις κοινές συνθήκες κυκλοφορίας. Για παράδειγμα, στις 6 το πρωί, ο αριθμός των περιστατικών ήταν ελάχιστος, κυμαινόμενος μεταξύ 0 και 30, καθώς η πρωινή κίνηση είναι συνήθως ελαφριά. Μεταξύ 7 και 8 το πρωί, ο αριθμός των περιστατικών αυξήθηκε σε 30 με 100, καταγράφοντας την αρχή της πρωινής ώρας αιχμής. Από τις 9 το πρωί έως τις 6 το απόγευμα, ο αριθμός των περιστατικών ήταν σημαντικά υψηλότερος, κυμαινόμενος από 150 έως 300, αντικατοπτρίζοντας τις συνθήκες αιχμής της κυκλοφορίας καθ' όλη τη διάρκεια της ημέρας. Μετά τις 6 το απόγευμα, ο αριθμός των περιστατικών άρχισε να μειώνεται, με μέτριο αριθμό περιστατικών μεταξύ 100 και 150 από τις 7 έως τις 8 το βράδυ, και κάτω από 100 περιστατικά από τις 9 έως τις 10 το βράδυ, αντιστοιχώντας στη μείωση της κυκλοφορίας καθώς η ημέρα πλησίαζε στο τέλος της.

Το μοντέλο *Python*, χρησιμοποιώντας τα *OR – Tools* της *Google*, συγκρίθηκε με τον *CP Optimizer* της *IBM*, υλοποιημένο σε *OPL*. Η απόδοση των δύο επιλυτών αξιολογήθηκε με βάση την ικανότητά τους να ελαχιστοποιούν τον συνολικό αριθμό των μοτοσικλετών και των πλατφορμών, ενώ εξυπηρετούν τουλάχιστον το 60% των περιστατικών κάθε ώρα. Σε 100 επαναλήψεις, η αποτελεσματικότητα κάθε επιλύτη μετρήθηκε από την ποιότητα της λύσης. Το μοντέλο Πψτηον παρουσίασε ισχυρή αρχική απόδοση, παράγοντας γρήγορα λύσεις και αξιοποιώντας τις τοπικές ευρετικές αναζητήσεις. Ωστόσο, σύμφωνα με επιστημονικές εργασίες, συχνά υπολείπεται σε πιο σύνθετες περιπτώσεις όπου ο αριθμός των μεταβλητών είναι μεγαλύτερος και οι περιορισμοί πιο αυστηροί.

Hour	Start_Moto	art_Platfor	Total_Moto	tal_Platfor	ved_Incide	tal_Incidents
6	70	0	70	0	7	14
7	0	0	42	0	25	50
8	0	0	42	0	27	55
9	0	36	42	36	78	156
10	0	41	42	77	119	238
11	0	0	42	77	113	226
12	0	0	42	77	98	196
13	28	3	70	80	150	300
14	0	0	0	80	75	151
15	0	0	0	80	80	160
16	28	0	28	80	100	201
17	0	0	28	80	108	217
18	0	0	28	80	81	162
19	0	0	28	80	54	109
20	0	0	28	80	67	134
21	0	0	28	80	39	79
22	0	0	28	80	45	90

Σχήμα 1.16: *Python studio*

Σε σύγκριση, ο *CP Optimizer* παρείχε σταθερά υψηλότερης ποιότητας λύσεις σε περίπου 72% των περιπτώσεων, ευθυγραμμισμένος με τα ευρήματα από παρόμοιες μελέτες. Οι ισχυρές δυνατότητες βελτιστοποίησης και η ικανότητα διαχείρισης σύνθετων περιορισμών του *CP Optimizer* τον κατέστησαν ιδιαίτερα αποτελεσματικό στη διαχείριση των υψηλών όγκων περιστατικών και των περίπλοκων προβλημάτων κατανομής πόρων που είναι τυπικά των ωρών αιχμής κυκλοφορίας. Αυτή η σύγκριση υπογραμμίζει τα πλεονεκτήματα του *CP Optimizer* στην παροχή αξιόπιστων, υψηλής ποιότητας λύσεων για σύνθετα σενάρια *VRP*, ενώ παράλληλα επισημαίνει την ικανότητα των *OR – Tools* να παράγουν γρήγορα καλές λύσεις για απλούστερες περιπτώσεις. Αυτά τα αποτελέσματα επικυρώνουν την αποτελεσματικότητα του *CP Optimizer* για απαιτητικά έργα βελτιστοποίησης, όπως αποδεικνύεται στη σχετική εργασία.



Σχήμα 1.17: *CPLEX results*

1.8 Συμπεράσματα

Συνοψίζοντας, η παρούσα διπλωματική εργασία εξερεύνησε τη χρήση αλγορίθμων μηχανικής μάθησης και προγραμματισμού περιορισμών για τη βελτιστοποίηση της ανάθεσης πόρων σε ασφαλιστικές εταιρείες. Μέσα από την ενσωμάτωση διαφόρων μοντέλων πρόβλεψης και τεχνικών βελτιστοποίησης, καταδείχθηκε πώς μπορούν να επιτευχθούν αποτελεσματικότερες και αποδοτικότερες λύσεις σε σύνθετα προβλήματα δρομολόγησης οχημάτων και διαχείρισης χωρητικότητας.

Επιπλέον, αποδείχθηκε ότι η χρήση της Γλώσσας Προγραμματισμού Βελτιστοποίησης (*OPL*) προσφέρει σημαντικά πλεονεκτήματα σε σχέση με το πλαίσιο *OR – Tools*, ιδίως σε ό,τι αφορά την απόδοση και την ευκολία συντήρησης των μοντέλων. Τα πειραματικά αποτελέσματα έδειξαν ότι το *OPL* παρέχει σταθερά υψηλότερης ποιότητας λύσεις για σύνθετα σενάρια, υπογραμμίζοντας την ανθεκτικότητα και την αποτελεσματικότητά του στη διαχείριση σύνθετων προβλημάτων βελτιστοποίησης.

Αυτή η έρευνα συμβάλλει στην συνεχιζόμενη ανάπτυξη και βελτίωση τεχνικών βελτιστοποίησης, προσφέροντας μια σταθερή βάση για μελλοντικές μελέτες και εφαρμογές σε διάφορους βιομηχανικούς τομείς. Η ενσωμάτωση προηγμένων αλγορίθμων και μεθοδολογιών μπορεί να οδηγήσει σε ακόμη πιο αποδοτικές λύσεις, υποστηρίζοντας τις επιχειρήσεις στην επίτευξη των στόχων τους με μεγαλύτερη ακρίβεια και αποδοτικότητα.

Κείμενο στα αγγλικά

Chapter 2

Introduction

Some of the most fundamental subjects in the field of computer science are machine learning, constraint programming and resource allocation problems. While in an undergraduate setting these would be studied separately, in a professional one they can be combined in order to efficiently solve a fairly complex problem. In this thesis, several important issues will be addressed. Drawing from the results of studies like [2] and [3], we will introduce a new approach to KPI prediction via machine learning techniques and implement it in the solution of a resource allocation problem. First, a thorough bibliographic research will be conducted which will cover all of the theoretical necessities of the thesis. This includes machine learning models and training techniques, but also constraint programming optimization theory. On the one hand we mostly concern ourselves with finding the most suitable machine learning model for a certain type of KPI prediction. On the other hand, we try to address the fairly complex problem of effective insurance driver shift assignment.

2.1 Motivating Illustration

The idea of machine learning arises with the fundamental question of whether machines can actually “think” like humans, meaning responding creatively, reasonably and consistently after being given a certain input. The aforementioned assumption, instantly creates functional questions as to how could this practically be done and what applications could exist in the industry. Let us give as example, the main problem this thesis will address and how machine learning has a vital role in solving it.

The problem involves an insurance company which has certain vehicles (resources) at its disposal and has the necessity of finding the best combination of the daily deployment of those vehicles. The latter, belong to two distinct categories: Motorcycles (M) and Platforms (P). The most effective way this problem can be optimised is to predict as accurately as possible the demand that can be met with the different combinations of M and P, and then choose the one that meets the company’s standards while minimising its expenses. A human expert might attempt to make such a prediction by drawing conclusions from various historical data about the average service time and the number of incidents a certain combination of vehicles resulted to in one hour. The most important problem is that the shifts of the vehicle drivers

are eight hours long, so every hour of each operational day would consist of a different (M, P) combination, making it extremely challenging to track. Certainly, one could do a statistical analysis of the data and then produce an acceptable prediction, but the question is whether there exists a more effective and less time-consuming technique.

Machine learning enables problem-solvers to approximate such complex tasks in the aforementioned manner. The idea would be to take advantage of an algorithm that can adapt to the peculiarities of the result that each historically known combination (M, P) produces and then predict the rest as accurately as possible. For example, if the needed key performance index to be predicted is the number of vehicles each (M, P) combination manages to serve in one hour, then there exists a scarce three-dimensional grid, constituting of the known historical data points. Each axis corresponds to the number of motorcycles, the number of platforms and the one of incidents served. The goal is to successfully fill in the rest of the grid without any major errors. Based on this prediction, one could call it a calculated guess, the shifts of the drivers and the (M, P) pairs can be drawn for every hour of every day.

Of course, our illustration in Figure 1.1 presents an idealised case in which the prediction is identical to the actual relation.

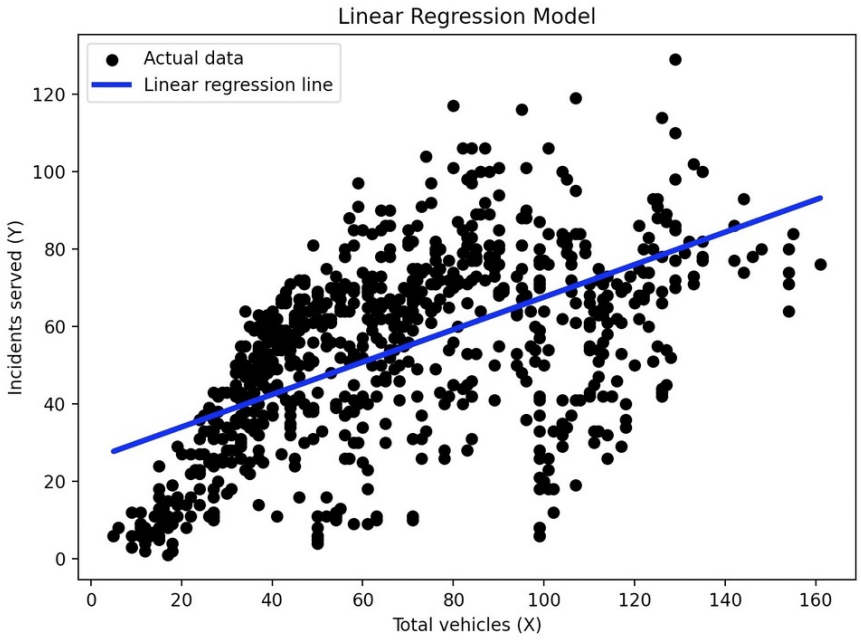


Figure 2.1

2.2 Our Contribution

In the industry of insurance company services, credibility is one of the most important values. When a large company has made certain guarantees to its customers and contractors, it is of vital importance that its services remain as close as possible to them. The reason behind the theme of this thesis is the very necessity of

a company to optimize its services and be able to guarantee timely results. With our deep examination of the effectiveness of the application of numerous machine learning models on KPI prediction, we find ways to optimize insurance company services.

This thesis delves into several state-of-the-art machine learning techniques, including linear regression, decision trees, random forests, and neural networks, to predict key performance indicators accurately. By comparing these models' performance, we identify the most suitable algorithms for different types of predictions required by the insurance industry. This rigorous evaluation not only enhances the accuracy of predictions but also provides insights into the strengths and weaknesses of each model, guiding future applications and research.

Besides the machine learning techniques studied and used in the first chapters, another problem will be addressed, which refers to the calculation of the best shifts for the insurance company drivers. As it will be explained in the thesis, the predictions of the machine learning models will be "plugged in" as input to an integer programming setting which will be calculating the most efficient shifts of any driver with certain constraints. The latter include getting 8-hour shifts or respecting holidays and are directly associated with the specific guidelines of the industry. Thus, a thorough bibliographic research on constraint programming and the applications of logic on computer science will be conducted. Then, in the last chapter of the thesis, we will propose ways of improving the credibility of the integer programming solution and thus the quality of the overall solution itself.

To ensure the robustness and reliability of the proposed solutions, this thesis also explores the impact of different data preprocessing techniques and feature engineering methods on the performance of machine learning models. By systematically analyzing and selecting the most relevant features, we enhance the models' predictive power and reduce computational complexity. This comprehensive approach ensures that the final models are both efficient and effective in real-world applications.

While the integration of machine learning and constraint programming offers significant advantages, it also presents several challenges. These include handling the complexity of real-world data, ensuring the scalability of the models, and addressing potential overfitting issues. Additionally, the dependency of the optimization results on the accuracy of machine learning predictions is a critical factor that needs careful consideration. To address these challenges, this thesis incorporates advanced techniques such as cross-validation, regularization, and hyperparameter tuning. These methods help to prevent overfitting, ensure model generalizability, and enhance the robustness of the optimization framework.

The introduction of advanced machine learning and optimization techniques provides a promising approach to solving complex resource allocation problems. This thesis aims to contribute to the ongoing research in this field by exploring the integration of these techniques and demonstrating their practical applications in the insurance industry. Moreover, by providing a detailed case study and empirical results, this research offers a reference for industry practitioners and researchers aiming to implement similar solutions in their contexts. The methodologies presented not only push the boundaries of current research but also pave the way for future innovations in the optimization of resource allocation in various industries.

2.3 Related Work

2.3.1 Machine Learning usage for KPI Prediction

Machine learning usage for KPI prediction is an extremely popular scientific field which has been vastly studied. Techniques used in this thesis have been discovered by scientists exploring machine learning applications. The most striking example, is the study for the accuracy of KPI prediction using machine learning techniques within the automotive industry [1].

The issue of KPI-related prediction was addressed by Manenti in 2009 [4], who also provided examples of how process performance monitoring is quickly shifting from reactive to predictive approaches. In his article, he provides a succinct review of mathematical approaches to the data reconciliation problem and suggests a fresh, potentially effective method to address it that is based on unstable detailed models. A novel method for automatically creating a dynamic predictive model to aid in more effective business process optimisation decisions was studied by Solomon and Litoiu in 2011 [5]. Via a simulation model, the prediction model establishes a correlation between the process variables and the KPIs. Several prediction models are used to estimate and forecast the input variables in the simulation model. The simulator then uses the anticipated inputs to determine the anticipated KPIs. Data-driven approaches require less prior physical and mathematical understanding of the process than model-based approaches do. The challenge of characterising unexpected behaviours makes model-based approaches frequently ineffective, why data-driven approaches have gained traction.

To return to our correlation, as EL Mazgualdi's study [1] showed the best performing models include algorithms that are comprised of ensemble learning techniques and neural networks. Much in accordance to our own findings. In that study, the authors try to predict the Overall Equipment Effectiveness (OEE) KPI, which is a widely used performance metric. Its computation gives managers the ability to pinpoint the primary losses that lower machine effectiveness and then make the required adjustments to rectify the situation. OEE was first presented by Nakajima in 1982 [6] and was a part of the total productive manufacturing (TPM) theory.

In the chase of effectiveness, El Mazgualdi and the other authors narrowed down the types of the models they studied to the most effective. Overall, four were studied: Support Vector Machines for Regression (SVRs), Random Forest, Extreme Gradient Boosting (XG-boost) and a deep learning neural network. The SVR is the only method of these not analyzed in the thesis. Formally, Support vector machine ([7], [8]) is a supervised machine learning algorithm mostly used in classification, but easily adopted for regression since it forms a generalization of the classification problem, in which the model returns a continuous- valued prediction. SVR can be defined as an optimization problem that relied on defining a convex e-insensitive loss function, then trying to minimize this function and find the flattest tube that contains most of the training instances. If the problem is linear, then the function f(x) can be described in the following form:

$$f(x) = \sum_{i=0}^{n} w_i x_i + b \tag{2.1}$$

While if we consider the case of a primal problem, we obtain the following minimization problem:

$$\begin{cases} \min_{w,b} \frac{\|w\|^2}{2} & \\ y_i - \langle w, x_i \rangle - b & \leq \epsilon, \quad \forall i \in \{1, \ldots, n\} \\ -(y_i - \langle w, x_i \rangle - b) & \leq \epsilon, \quad \forall i \in \{1, \ldots, n\} \end{cases} \tag{2.2}$$

Two new variables, ξ and ξ^*, are introduced to tackle this optimisation problem, which is frequently unsolvable. Next, we have the newly created problem:

$$\begin{cases} \min_{w,b} \frac{\|w\|^2}{2} + C \sum_{i=0}^{n} \xi_i + \xi_i^* & \\ y_i - \langle w, x_i \rangle - b + \xi & \leq \epsilon, \quad \forall i \in \{1, \ldots, n\} \\ -(y_i - \langle w, x_i \rangle - b + \xi^*) & \leq \epsilon, \quad \forall i \in \{1, \ldots, n\} \\ \xi_i^*, \xi_i & \geq 0, \quad \forall i \in \{1, \ldots, n\} \end{cases} \tag{2.3}$$

Although promising, this method was not properly addressed in this thesis due to its high parameter sensitivity, computational expenses and challenging interpretability. While they can make accurate predictions, understanding how the features influence the prediction can be difficult, especially with non-linear kernels. Finally and possibly most importantly, support vectors have a binary nature. While this is efficient, it also means that they might miss some information from the data that could be potentially useful, especially if the data is not well-represented like in our case.

Key to the study were the three core parameters impacting OEE: availability, performance, and quality. However, given the consistent 100% quality rate across the dataset, the study focused on the remaining two factors. Through experienced-based feature selection and engineering, a concise set of input variables was identified. These included setups, breakdowns (limited to planned maintenance for predictability), the number of orders per shift, mean wire length, and the number of terminals and seals per order, along with a target variable reflecting the order's complexity. All things considered, the fact that the Random Forest Regressor was considered one of the best candidate models in this study, certainly is a reassurance.

Another study conducted by Joel Shodamola in 2020 [3], regarding the usage of machine learning models for KPI maximization in emerging networks has adopted a similar approach. It presents a novel machine learning-based framework aimed at maximizing Key Performance Indicators (KPIs) in cellular networks through the optimization of Configuration and Optimization Parameters (COPs). The proposed framework consists of three main components: data generation using a cellular network simulator, KPI prediction through various machine learning algorithms, and the identification of optimal COP combinations via GA. Therefore, the relative study compares the performance of five machine learning models - Linear Regression, K-Nearest Neighbor (KNN), Extreme Gradient Boosting (XGBoost), Categorical Boosting (CatBoost), and Deep Neural Network (DNN) - with CatBoost emerging as the most effective in predicting KPI behavior.

A similar setup to ours was followed with simulations generating relevant data and then tests being conducted on that data. As the validation function, machine learning prediction performances were assessed in terms of Root Mean Square Error (RMSE). As expected, Linear Regression exhibited the lowest performance due to its constraints in matching non-linear data and forecasting non-training set data. Because CatBoost precisely categorised each attribute and minimises overfitting, it

performed the best. Another intriguing finding was the ML models' strong performance even with 10% of training data. CatBoost is an open-source machine learning algorithm developed by Yandex researchers and engineers, and is designed to work with categorical data directly, without the need for extensive data preprocessing required by other machine learning algorithms. Being part of the gradient boosting family of algorithms, it is very close to the ones already analyzed.

Due to their problem-solving nature, machine learning algorithms have a vast number of applications. Besides the above, in an article titled "Machine Learning Techniques to Predict Reactionary Delays and Other Associated Key Performance Indicators on British Railway Network" [2] the significant issue of reactionary delays within the British railway system is being addressed. These delays, which originate from a primary source and propagate throughout train journeys, result from complex non-linear interactions between various spatio-temporal variables, causing further disruptions across the network. The paper proposes a novel approach utilizing advanced machine learning techniques, particularly focusing on XGBoost (Extreme Gradient Boosting) and Deep Neural Networks (DNN), to predict key performance indicators (KPIs) such as reactionary arrival delay, reactionary departure delay, dwell time, and travel time.

The main problem described in the paper is the increasing reactionary delay minutes per year in the British rail industry, despite the rail network being one of the oldest and most extensive in the world. This issue has significant implications for the rapidly growing commuter population, leading to increased delay compensations claims and financial losses for Train Operating Companies (TOCs). The research aims to improve upon existing delay propagation prediction systems by offering a more accurate and reliable framework that can aid TOCs in making better delay-mitigating decisions.

The roles of XGBoost and Deep Neural Network algorithms in this project are crucial. These state-of-the-art machine learning models are employed to predict the aforementioned KPIs with greater accuracy compared to existing systems. Specifically, the paper outlines how these models are individually calibrated and optimized to predict KPIs for specific time horizons or "steps" throughout a train's journey. This detailed prediction capability allows for a train-centered approach, enhancing the ability to provide real-time predictions and improve the overall performance of the UK railway delay prediction system, notably outperforming the current system known as Darwin.

So overall, the results of the study demonstrate the effectiveness of the XGBoost and DNN models in predicting the specified KPIs with high accuracy, outperforming the existing delay prediction system known as Darwin. Interestingly, the models achieved lower root mean square error (RMSE) values for arrival and departure delay predictions compared to Darwin, indicating a significant improvement in prediction accuracy. Furthermore, the analysis of feature importance using the SHAP (SHapley Additive exPlanations) framework revealed that the departure delay from the previous station was a critical predictor for both arrival and departure delays, underscoring the interconnected nature of delays within the rail network.

The last study also proved the right direction of our way of thinking and approach to the solution. While the Neural Networks might offer quite more accurate predictions,

they can often be overly complicated and take more time or resources to execute. A strong indication of the aforementioned claims are the figures provided by [2] and presented below.

	DNN				XGBoost			
Step	RMSE	r^2	ADTAE	MAAPE	RMSE	r^2	ADTAE	MAAPE
1	1.09	0.64	0.55	0.82	1.07	0.65	0.51	0.81
2	1.18	0.58	0.62	0.81	1.13	0.61	0.55	0.83
3	1.06	0.65	0.52	0.91	1.06	0.65	0.48	0.89
4	0.99	0.68	0.45	0.96	1.03	0.65	0.43	0.98
5	0.96	0.69	0.40	1.03	0.97	0.68	0.40	1.03
6	0.99	0.66	0.37	1.09	0.94	0.70	0.37	1.10
7	0.86	0.74	0.32	1.14	0.91	0.71	0.34	1.15
8	0.84	0.74	0.30	1.20	0.80	0.77	0.29	1.19
9	0.81	0.74	0.29	1.22	0.72	0.80	0.26	1.23
10	0.86	0.70	0.26	1.26	0.78	0.75	0.23	1.25

Figure 2.2: Model performances on the dwell time KPI

All in all, the research underscores the potential of advanced machine learning techniques to enhance the prediction of train delays, offering a promising avenue for improving the operational efficiency and reliability of railway systems. The application of XGBoost and DNN models not only offers a method to better understand and predict delay propagation but also highlights the importance of leveraging big data and machine learning to address complex challenges in public transportation networks.

Model	DNN Training Time (s)	XGBoost Training Time (s)
1-step	137.28	86.46
2-step	140.21	83.37
3-step	140.67	83.34
4-step	141.06	86.99
5-step	140.09	83.66
6-step	139.70	84.52
7-step	142.22	82.94
8-step	142.97	83.48
9-step	135.80	81.89
10-step	141.12	83.54

Figure 2.3: Model running times

Finally, another striking example of a scientific paper with similar problem solving approach to ours is named "A Practical Model for Traffic Forecasting based on Big Data Machine-learning and Network KPIs" and presents a comprehensive approach to traffic forecasting within mobile networks, particularly focusing on the 5G self-organizing network (SON) environment. Most existing models rely heavily on historical traffic data, with limited use of other network Key Performance Indicators (KPIs). The authors propose a novel forecasting model that integrates big data machine learning techniques and network KPIs, aiming to accurately forecast traffic for GSM, 3G, and 4G cell types over both short and long terms. This model was evaluated using real dataset KPIs from over 6000 network cells during 2016 and 2017.

The paper begins by highlighting the complexity of traffic patterns in cellular networks, which vary significantly across different times and geographical locations. This complexity makes traffic forecasting a challenging but crucial task. Traditional machine learning approaches to traffic forecasting have been cell-specific and limited to short-term predictions, with a narrow understanding of the factors influencing traffic variations. In contrast, the proposed model in this study aims to provide a more flexible and accurate forecasting tool that considers various KPIs, thereby enhancing the network's spectral efficiency and reducing power consumption.

The forecasting process is divided into two main steps: analyzing the dataset to understand the relationship between KPIs and future traffic, and then applying ML algorithms for actual traffic forecasting. The paper details the use of Bayesian networks to analyze the probabilistic relationships among different KPIs and traffic, identifying which KPIs are most important for accurate predictions. This methodology ensures that the forecasting model focuses only on significant parameters, improving its efficiency and ease of use.

The results showcase the effectiveness of the proposed model, with various ML algorithms (Auto Regressive, Neural Networks, and Gaussian Process) tested for accuracy. The Gaussian Process (GP) algorithm, in particular, demonstrated superior performance in handling rapid traffic changes, showcasing the advantage of integrating crucial KPIs into the forecasting model. The paper concludes that the proposed approach provides a more accurate, stable, and practical solution for traffic forecasting in cellular networks, outperforming existing models by leveraging big data analytics and machine learning to address the dynamic and complex nature of network traffic.

2.3.2 Constraint Programming: OPL vs OR-Tools

In the comparative study "Google vs IBM: A Constraint Solving Challenge on the Job-Shop Scheduling Problem," [9] the performance of IBM's CP Optimizer and Google's OR-Tools was evaluated on a set of classic and large-scale job-shop scheduling benchmarks. The results demonstrated that CP Optimizer consistently outperformed OR-Tools in terms of solution quality and robustness across different problem instances. Specifically, CP Optimizer was able to find optimal solutions more frequently and handle larger, more complex instances more effectively than OR-Tools. This work highlighted the strength of CP Optimizer's sophisticated search strategies and its ability to maintain high performance even under stringent constraints.

In this thesis, a similar comparative analysis between IBM's CP Optimizer and Google's OR-Tools was conducted, but focused on the Vehicle Routing Problem (VRP). The VRP shares similarities with job-shop scheduling in terms of its combinatorial complexity and the necessity for efficient resource allocation. By applying both solvers to a variety of VRP instances, we aimed to assess their performance in terms of solution quality, solving time, and scalability. Consistent with the findings in the job-shop scheduling study, CP Optimizer generally produced higher quality solutions, particularly for larger and more complex VRP instances, confirming its robustness and effectiveness.

Furthermore, our study revealed that OR-Tools, while competitive, particularly in smaller instances or with multi-core configurations, often required more computational time to reach solutions of comparable quality to those found by CP Optimizer. This aligns with the job-shop scheduling findings, where OR-Tools showed significant improvements in multi-core environments but still lagged behind CP Optimizer in terms of solution optimality and consistency. These observations underscore the versatility and advanced optimization capabilities of CP Optimizer, especially for industrial-scale problems.

Adding to the above, the work done by Thibaut Cuvelier and colleagues from Google Paris in the paper "OR-Tools' Vehicle Routing Solver: a Generic Constraint-Programming Solver with Heuristic Search for Routing Problems" [10], explores the capabilities of Google's OR-Tools, particularly focusing on its application to vehicle routing problems (VRPs). OR-Tools is an open-source optimization library developed to solve various optimization problems, including VRPs. The paper details the solver's development, which began in 2008, and its release as an open-source tool in 2015. It includes multiple solvers, such as CP* and CP-SAT for constraint programming, and GLOP and PDLP for linear programming, showcasing its versatility and robust

performance, especially for large-scale industrial applications.

The vehicle routing solver in OR-Tools is emphasized for its ability to handle complex VRPs with multiple constraints, such as vehicle capacities, time windows, traffic considerations, and driver breaks. The solver employs a three-phase approach: first-solution heuristics to generate initial routes, local search techniques to refine these routes, and metaheuristics like simulated annealing and tabu search to escape local optima. The final optimization phase uses a constraint programming engine to prove the optimality of the best solution or improve it further. This structured methodology ensures that OR-Tools can efficiently address the complexities of real-world VRPs, providing a high-level modeling API accessible in several programming languages.

The results from our work reinforce the conclusions drawn in the paper, emphasizing the superior performance of CP Optimizer for complex optimization problems like VRP. Both studies illustrate that while OR-Tools is a powerful and flexible tool, particularly advantageous for smaller or less complex instances, CP Optimizer remains the preferred choice for scenarios demanding high solution quality and reliability under rigorous constraints. This comparison not only validates the robustness of CP Optimizer across different domains but also highlights the critical need for tailored solver selection based on specific problem characteristics and computational requirements. Consistent with the findings in the paper, this thesis demonstrates that OR-Tools is highly effective in generating initial solutions quickly and improving them through local search and heuristics. However, CP Optimizer frequently delivers higher quality solutions for more complex instances, reflecting its robust optimization capabilities. This comparison highlights the strengths and limitations of each solver, demonstrating their applicability to solving VRPs under different constraints and requirements, thereby validating the conclusions drawn about OR-Tools' effectiveness and flexibility in the paper.

Chapter 3

Machine Learning Basics

Machine learning refers to the development of statistical algorithms, with the goal of performing tasks without explicit programming. It involves learning from known data and generalising to unknown, thus making calculated predictions. The term machine learning is often linked to Cornell University's Dr. Frank Rosenblatt [11], who was instrumental in developing a machine that could recognise letters of the alphabet. Since then, machine learning models have attracted an overwhelming interest of both academics and professionals, being on the verge of transforming their operational paradigms (e.g.[12], [13], [14]). In this chapter, we will present some of the basic machine learning models and a variety of ways they can be used to address complex resource allocation problems.

3.1 Linear Regression Models

Probably one of the simplest forms of supervised learning is linear regression, a rather useful tool for predicting a quantitive response. It is a widely used statistical method which separates the problem variables into a dependant and one or more independent ones. The core idea behind it is to fit a linear equation to the observed data, which can then be used to predict outcomes for new, unseen data. It is a fundamental algorithm useful not only for predicting the behaviour of data but also for understanding the underlying patterns and relationships within complex data structures Maulud [2020]. Despite its simplicity, it constitutes the basis for understanding more complex principles and algorithms regarding machine learning.

3.1.1 Simple Linear Regression

This technique truly lives up to its name. It constitutes of one response Y (dependant variable) and a single predictor X (independent variable), assuming that the relationship between the two is approximated linearly. Thus, the mathematic model is "simple" itself:

$$Y = b_0 + b_1 X \tag{3.1}$$

In a simplified version of our previous example, Y can be the the number of incidents served in one hour while X the overall number of available vehicles from the insurance company. Then Y can be regressed onto X according to the relation (1.1).

With b_0 and b_1 representing the intercept and the slope respectively. It should be noted that these are initially unknown. So basically the whole point of the training becomes to determine these two numbers. Based on certain observation pairs

$$(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n) \tag{3.2}$$

Each consisting of a measurement of X and the respective value of Y, the pair (b_0, b_1) is gradually determined through measuring the "closeness" of the occurring line from the given data points James [2023]. The way this "closeness" can be computed can be found through various mathematical methods, the most important of which will be analysed later. The more closeness is minimised, the better the models fits the training data. One of the most famous mathematical methods used to optimise this problem is called Ordinary Least Squares (OLS) and it works by defining

$$e_i = y_i - \bar{y}_i, \tag{3.3}$$

As the ith residual of the data set. If there are n training data points, then there will exist n residuals, each of which affecting the final prediction equally. Then, the Residual Sum of Squares is defined as:

$$RSS = e_1^2 + e_2^2 + \ldots + e_n^2 = \sum (y_i - \bar{y}_i)^2 = \sum (y_i - b_0 - b_1 X_i)^2 \tag{3.4}$$

If we further define the sample means as

$$\bar{Y} = \frac{1}{n} \sum y_i, \quad \bar{X} = \frac{1}{n} \sum X_i \tag{3.5}$$

Then using some basic calculus, the coefficients that minimise the *RSS* can be determined through the following equations [15]:

$$b_1 = \frac{\sum (X_i - \bar{X})(y_i - \bar{Y})}{\sum (X_i - \bar{X})^2} \tag{3.6}$$

and

$$b_0 = \bar{Y} - b_1 \bar{X} \tag{3.7}$$

It should be noted that an independent catch-all error term ϵ will always be necessary mainly because of the high probability of the true relation between X and Y not being linear, measurement errors and the chance that Y is affected by other variables beyond X. So the linear mathematical relationship can be finalised as follows:

$$Y = b_0 + b_1 X + \epsilon \tag{3.8}$$

This is because it is only natural for every experimentalist to have the desire to know the extent which the model has fitted the training data. Depending on the problem, if the model "learns too much" from the data, a term defined as *overfitting*, might cause unwanted results. The most common ways of measuring this extent are the Residual Standard Error (*RSE*) and the R^2 score.

3.1.2 Residual Standard Error

The *RSE* is an estimate of the standard deviation of e. It quantifies how much the data points deviate from the fitted regression line, and therefore provides a measure of the model's fit. Furthermore, it is particularly useful in comparing different model's performance on the same training dataset. Naturally, lower values indicate a better fit while higher values a worse one, instantly quantifying the size of the errors each models makes. It is computed using the formula [16]:

$$RSE = \sqrt{\frac{1}{n-2}RSS} = \sqrt{\frac{1}{n-2}\sum(y_i - \bar{y})^2} \tag{3.9}$$

Note that *RSS* is defined in 3.14. In the example shown in figure 2.1, the *RSE* value is 20.42, which indicates that the actual number of incidents deviates from the one predicted by approximately 20.42 incidents on average. Obviously, whether or not this number is an acceptable deviation from the real data or not depends entirely on the problem context.

3.1.3 R^2 Score

This measurement, just like the *RSE* provides a solid representation of the model's fit to the data. It basically bypasses the *RSE*'s limitation of being measured in units of Y. Because it represents the proportion of the variance for a dependent variable that's explained by an independent one, it takes values between 0 and 1 and is independent of the scale of Y. It indicates how much the model explains the variability of the response data around its mean. A score near 1 means that a large proportion of the variability of the response is explained by the regression, and as it decreases, so does the respective proportion. Mathematically, R^2 is defined by [16]:

$$R^2 = 1 - \frac{RSS}{TSS} \tag{3.10}$$

where

$$TSS = \sum(y_i - \bar{y})^2 \tag{3.11}$$

Back to our example shown in figure 2.1, after some experimenting the R^2 score turned out to be 0.287, which is really not impressive. Besides its many practical uses, simple linear regression does not have any major ability to adapt to the unique characteristics a more complex dataset may have. Consequently, there is a need of a type of machine learning model with better adaptation skills.

3.1.4 Multiple Linear Regression

The four basic assumptions made with simple linear regression must also hold for multiple linear regression, as the multiple linear regression model is based on the same basis as simple linear regression. However, a new set of concepts needs to be introduced in addition to the ones that have already been discussed and are still relevant for simple linear regression. The scenario with two predictor variables and one outcome variable will be the focus of this discussion. The data can be visualised using a three-dimensional figure with a total of three variables. The same

concepts apply to increasingly complex models, but they are more challenging to visualise.

The aforementioned skills occur when more than one predictors exist in the model's structure. In real world problems, there are numerous occasions where the outcome will be depending on more than one variable. While the multiple linear regression model is built on the same foundation as the simple linear regression, there are a few new concepts that need to be introduced. We will primarily concentrate on the occasion where there are two predictor variables and one outcome. Thus, a three dimensional grid is formulated like the one discussed in section 2.1, in order to visualise the data. The equation for the regression model given below, represents a flat plane in the three dimensions:

$$Y = b_0 + b_1X_1 + b_2X_2 + \cdots + \beta_nX_n + \epsilon \tag{3.12}$$

Where obviously, Y is the outcome, X_1, X_2 the two predictor variables, and b_0, b_1, b_2 the coefficients we are looking for. Once again, ϵ is a catch-all error term. It is worth to be noted that since the relation between the outcome and the predictor variables is linear, the grid formed will be always a flat plane with no curves or bendings Keith, Marill [2004]. The plane that fits best the training data can be found using the least squares approach that was described in subsection 3.1.1. The regression coefficients will be determined using the formula:

$$\beta = (X^TX)^{-1}X^TY \tag{3.13}$$

where β is the vector of coefficients (including β_1, β_2,...,β_k for k coefficients) X is the matrix of input features (including a column of ones for the intercept), Y is the vector of the observed values, and $(X^TX)^{-1}X^TY$ represents the matrix operation to solve for the vector of coefficients. Similarly, the *RSS* is defined as follows:

$$RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.14}$$

Back to the insurance company example, assuming that the number of incidents served is dependant only on the total number of vehicles is an oversimplification of the problem and thus, probably not a viable solution. Because it is a real world application and there are two different variables influencing the outcome, there is a necessity of a surrogate (prediction) model that uses the equation 3.12 with two predictors. This way, while it might not give as effective a prediction as other models that we will discuss later may provide, the problem solution will be approximated much better [17]. The goal is to find the coefficients that minimize 3.14. The mathematical expression for the two-predictor model, arises from 3.12:

$$Y = b_0 + b_1X_1 + b_2X_2 + \epsilon \tag{3.15}$$

Using this equation, we train our new model in order for it to generate the following plane:

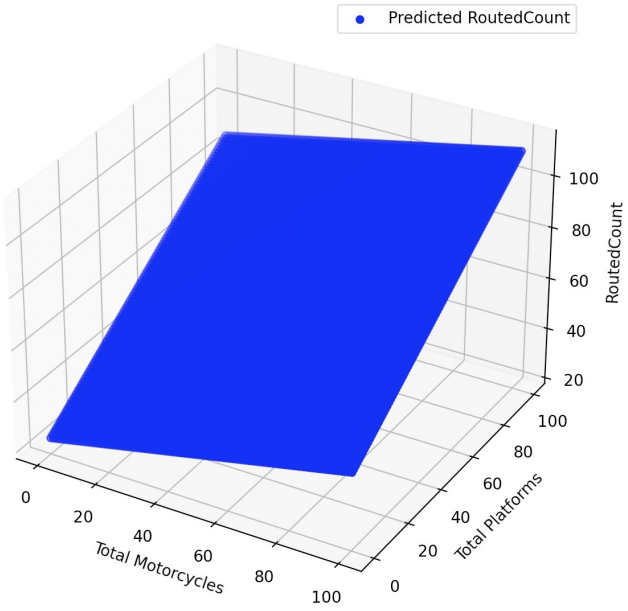


Figure 3.1

Which is a much improved representation of how one would imagine the actual correlation between the number of Motorcycles, Platforms and Incidents Served (RoutedCount). As we would expect, when (M, P) approaches $(0, 0)$ the number of incidents served tends to 0 as well, and when (M, P) approaches $(100, 100)$ the respective RoutedCount takes some of its highest values. Multiple linear regression, offers new information regarding how the number of platforms (P) affects more the RoutedCount value than the number of Motorcycles (M).

3.2 Non-Linear Regression Models

Non-linear regression machine learning models are powerful tools for understanding and predicting outcomes where the relationship between independent and dependent variables is complex and does not follow a straight line. Their ability to from curvilinear predictions makes them highly adaptable to a wide range of data structures, and often more appealing than their linear counterparts. Therefore, this kind of models can fit the data more accurately and provide more precise predictions in scenarios such as exponential growth, logarithmic growth, or other non-linear patterns. However, the increased complexity of such models is not always beneficial, as it may lead to challenges in model interpretation, higher computational costs, and a greater risk of overfitting the model to the training data. In the next subsections, we will discuss different non-linear regression models and how they can be utilised to approximate complex problems better.

3.2.1 Polynomial Regression

The main idea behind the reason why polynomial regression has proved to be an extremely effective tool for regression models originates from a theorem Karl Weierstrass wrote in 1885, later named as the Weierstrass Approximation Theorem [Davis, 1975:Chapter VI]. It implies that any continuous function on a finite interval can be approximated arbitrarily closely by a polynomial. This involves consolidating any residual terms from expanding the unknown model function using a Taylor series into the error component. While enhancing the polynomial's order can refine this approximation, the cost is an increase in the number of unknown parameters and might induce fluctuations across the data points. The computational formula behind the model is:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \cdots + \beta_{n1} X_1^n + \beta_{n+1} X_2 + \beta_{n+2} X_2^2 + \cdots + \beta_m X_2^n + \cdots + \beta_k X_p^n + \epsilon \quad (3.16)$$

While theoretically, it's feasible to model a polynomial up to the $n-1$ degree several practical challenges emerge when k (the degree of the polynomial) is significantly high. Particularly when it exceeds roughly 6, the regression matrix X related to the model becomes prone to numerical instability or becomes "ill-conditioned" [18]. The simplest polynomial is represented by a second- degree or quadratic equation of the form:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon \quad (3.17)$$

One disadvantage of polynomial regression is that the polynomial whose graph approximates the most the problem's curve, must be found in the model training process. But now, there is the option of experimenting with the polynomial's powers as well, adding a new complexity layer to the solution. This experimentation is essential because a model with a degree that is too low might not capture the true relationship (underfitting), while a model with a degree that is too high might learn the noise in the data rather than the actual signal (overfitting). Techniques like *RSE* and the R^2 score described in 3.1.2 and 3.1.3, are used to determine whether a certain polynomial is capable of approximating the problem correctly or not.

3.2.2 k-Nearest Neighbors (kNN)

The k-Nearest-Neighbours (kNN) is a non-parametric classification method, which is simple but effective in numerous cases. Going back to the training datapoints, the main idea behind this approach is for every point p that is to be predicted, to gather the k nearest known datapoints and calculate their average value. These k nearest datapoints form the neighbourhood of p. Different metrics may be used to define the term "nearest", with the most frequently used one being the Euclidean Distance between the points. Nevertheless, employing the kNN algorithm necessitates the selection of an optimal value for k with the algorithm's effectiveness being highly contingent on this choice. Essentially, the performance and bias of the kNN method are influenced by the selection of k. The most common way of determining the exact value of k, is through a trial and error process during which different values are tested [19]. Although there does not exist a formal mathematical formula for the *kNN* model, intuitively the process can be expressed by the following equation:

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i \quad (3.18)$$

Here, the predicted value for a new data point is calculated by averaging the outcomes of the closest k data points to it. In classification models, *kNN* has a comparatively high cost of classifying new instances. This is due to the fact almost all computation takes place at classification time rather than when the training examples are first encountered. Consequently, this prohibits it from being used in research areas where dynamic classification is needed for large sets of data.

3.2.3 Decision Trees

A decision tree, having its origin in machine learning theory, is an efficient tool for the solution of classification and regression problems. Contrary to other classification techniques that apply a collection of characteristics for categorization in a single step, decision trees use a hierarchical approach, similar to a tree structure. This structure consists of nodes categorised as internal, terminal, and a root that contains the complete dataset. We can imagine the internal nodes as decision stations, and the terminal as the leafs of the tree. At every decision node a binary choice is made, essentially separating one or more classes from the others. The basic idea behind such models is to break down a complex problem into several simpler ones and following a top-down approach reach a conclusion that is easier to unravel [20]. In such processes, features of data (i.e., bands) are predictor variables whereas the class to be mapped is referred to as the target variable. The difference between decision tree regression and classification depends on whether the target variables are continuous or discrete. Visually, a decision tree would look like this:

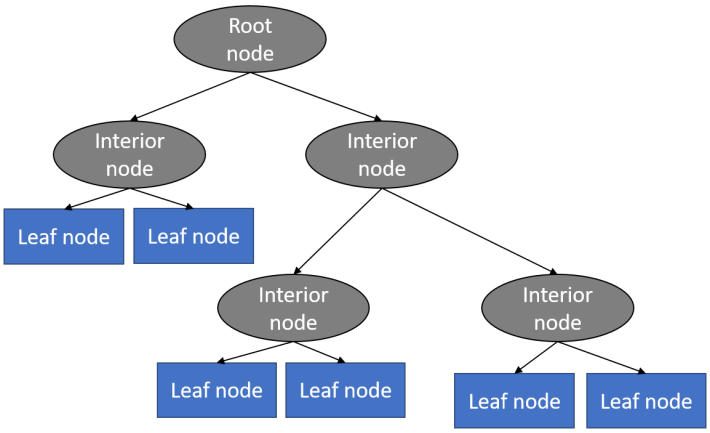


Figure 3.2

The construction of a binary regression tree is based on binary recursive partitioning, which is an iterative process that splits the data into partitions. The structure of the tree is determined by all the training samples. The algorithm then breaks the data using every possible binary split and selects the split that partitions the data into two parts such that it minimizes the sum of the squared deviations from the mean in the separate parts. The splitting process is then applied to each of the new branches recursively until a leaf is reached. The decision to split at each node is made with the goal of partitioning the data into groups of similar values of the target variable as much as possible, this way leading to more precise predictions. In other words, the reduction in every node's MSE must be maximised during the split decision.

Mathematically, this can be expressed as:

$$\text{MSE}_{\text{node}} = \frac{1}{N_{\text{node}}} \sum_{i \in \text{node}} (y_i - \bar{y}_{\text{node}})^2 \tag{3.19}$$

Where MSE_{node} represents the mean squared error for a given node, N_{node} is the number of samples in the node, y_i is the actual value of sample i and $\bar{y}_{\text{node}}$ is the mean value of the target variable in the node.

3.2.4 Random Forest

The random forest model inherently succeeds the decision tree. It is worth to be noted that just like most of this thesis, in this paragraph we will focus on the regression aspect of the model rather than the classification. The algorithm consists of a collection (forest) of decision tree models with the common goal of improving the prediction precision and limit over-fitting chances. So, assuming we have a tree predictor $h(\mathbf{x}; \partial_k)$ where $\mathbf{x}$ represents the observed input vector of length p with associated random vector $\mathbf{X}$ and the ∂_k are independent and identically distributed (iid) random vectors [21]. As always, we assume that the training data is independently drawn from the joint distribution $(\mathbf{X}, Y)$ and comprises n tuples $(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n)$, just like 3.2 shows. Calculating the unweighted average over the collection

$$H(\mathbf{x}) = \frac{1}{K} \sum_{k=1}^{K} h(\mathbf{x}; \partial_k) \tag{3.20}$$

Thus, as $k \to \infty$, the Law of Large Numbers ensures:

$$E_{\mathbf{X},Y}(Y - \hat{h}(\mathbf{X}))^2 \to E_{\mathbf{X},Y}(Y - E_{\partial} h(\mathbf{X}; \partial_k))^2 \tag{3.21}$$

Which is a convergence implying that random forests do not overfit, as the quantity on the right is the generalization error. Additionally, in order to have precise random forest predictions, there is a necessity of low prediction errors of the individual trees, and low correlation between residuals that belong to different trees in the forest. The aforementioned claims can be proven if we define the average prediction error for an individual tree $h(\mathbf{x}; \partial_k)$, as:

$$PE_t^* = E_{\partial} E_{\mathbf{X},Y}(Y - h(\mathbf{X}; \partial_k)) \tag{3.22}$$

Assuming further that for all ∂ the tree is unbiased, i.e., $EY = E_{\mathbf{X}} h(\mathbf{X}; \partial_k)$, then:

$$PE_f^* \leq \hat{\rho} PE_t^* \tag{3.23}$$

where $\hat{\rho}$ is the weighted correlation between residuals $Y - h(\mathbf{X}; \partial)$ and $Y - h(\mathbf{X}; \partial')$, for independent ∂, ∂'. Finally, random forest models do not just come in handy when an accurate prediction is required. Among other examples, they can accurately estimate test sets' prediction errors and distinguish forests from so-called black-box predictors (e.g., neural nets). Below, there is a graphic illustration of a random forest model.

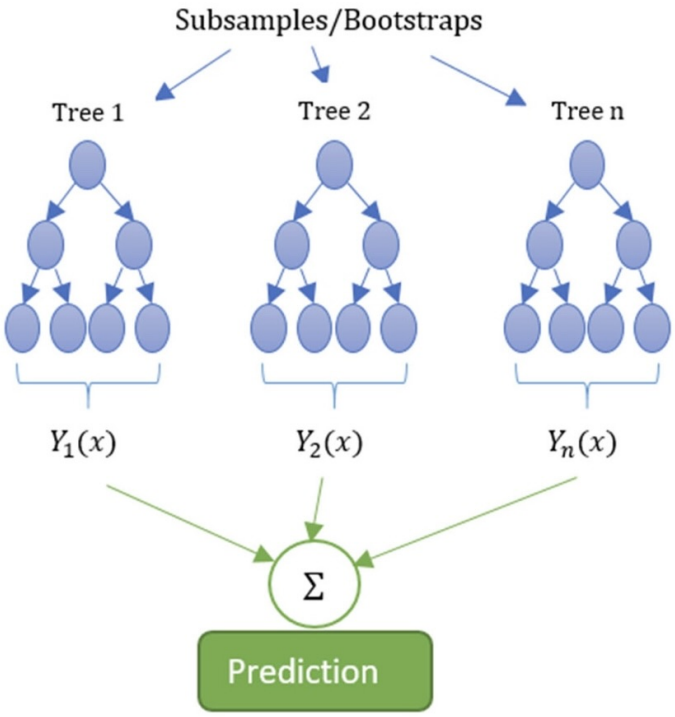


Figure 3.3: Random Forest graphically [1]

3.2.5 Gradient Boosting

Gradient boosting is a boosting algorithm for regression. It involves training multiple submodels sequentially, usually decision trees, in a way that each one corrects the mistakes of the previous. Each new model in the sequence focuses on accurately predicting the residuals or errors of the prior models and then adds weights to them in order to produce its final result. This way, the overall model's accuracy is improved incrementally with each step. Gradient Boosting Regression is extremely flexible in optimizing different loss functions and handling diverse types of data. It often compares favourably to aforementioned models, duo to its effectiveness in dealing with complex and non-linear datasets [22]. This technique can surpass in accuracy traditional machine learning models, especially when its parameters such as such as the number of boosting stages, learning rate, and depth of trees, are tuned carefully. So, mathematically, given a training dataset $D = (\mathbf{x}_i, y_i)_1^N$ gradient boosting algorithm's goal is to find an approximation, $\hat{F}(\mathbf{x})$, of the function $F^*(\mathbf{x})$, which maps instances $\mathbf{x}$ to their output values y. This happens by minimizing the expected value of a given loss function $L(y, F(\mathbf{x}))$. Furthermore the aforementioned "weighted" combination of the previous submodels, can be expressed as:

$$F_m(x) = F_{m-1}(x) + \rho_m h_m(x), \tag{3.24}$$

where ρ_m is the weight of the m^{th} function, $h_m(\mathbf{x})$, which in most cases, practically represents a decision tree. The iterative process begins by obtaining a constant approximation of $F^*(\mathbf{x})$, as:

$$F_0(x) = \text{argmin}_a \sum_{i=1}^{N} L(y_i, a) \tag{3.25}$$

the values expected to be minimized by subsequent models are:

$$(\rho_m, h_m(x)) = \text{argmin}_{(\rho,h)} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \rho h(x_i)) \tag{3.26}$$

However, if proper regularisation is missing from the iterative process, the model is in danger of suffering from overfitting. In some cases of loss functions, if the model fits the pseudo-residuals ideally then they become zero and the process terminates prematurely. So naturally, there is a need of regularizing this gradient boosting process. With v = (0, 0.1], shrinkage can be applied via the equation:

$$F_m(x) = F_{m-1}(x) + \upsilon\rho_m h_m(x) \tag{3.27}$$

where v is usually set to 0.1. Moreover, techniques like limiting the complexity of the models used can be applied, in order to achieve regularisation. In the case of decision trees, the most common way to achieve this would be to limit the depth of each tree.

3.2.6 Gaussian Process

This is a Bayesian, non-parametric approach to regression, that uses probabilistic techniques to predict the distribution of unknown functions. The model practically models the underlying function as a Gaussian Process, meaning a collection of random variables, any finite number of which have a joint Gaussian distribution. This way, the model can not only predict the expected value of the function at a given point, but further estimate the probability of that prediction being wrong. A statistical model taken from the literature on spatial statistics is the GASP model. The kriging model for a given correlation function selection corresponds to the Gaussian Process model [23]. The deterministic output response is viewed by the GASP model as a multivariate normal, or a realisation of a random stochastic process. The output response is represented as an $n \times 1$ data vector y(x) with mean $\mu 1_n$ (also $n \times 1$) and covariance

$$\text{Var}(y) = \sigma^2 R(X, \partial) \tag{3.28}$$

where the correlation matrix $R(X, \partial)$ is $n \times n$. A function of the design space, design points, and unknown thetas is the correlation matrix $R(X, \partial)$. This correlation function can be expressed in a number of ways, but the version we'll choose is as follows:

$$R_{ij}(X, \partial) = \exp\left(-\sum_k \partial_k (x_{ik} - x_{jk})^2\right) \tag{3.29}$$

where $\partial_k \geq 0$. Furthermore, the correlation is 1.0 across the range of the $k_t h$ factor, if $\partial_k = 0$. It occurs naturally that large ∂_k values correspond to low correlation and a bumpy fitted surface in the respective direction. Utilizing the fact that the maximum likelihood estimates can be represented as $\hat{\mu}, \hat{\sigma}$ and $\hat{\partial}$ the prediction equation can take the form:

$$\hat{y}(x) = \hat{\mu} + r'(x, \hat{\partial}) R^{-1}(X, \hat{\partial})(y - \hat{\mu} 1_n) \tag{3.30}$$

where $r_i(x, \hat{\partial})$ is an n$\times$1 vector of estimated correlations between the data's observations, y(x), and the unobserved y(x) with a new value for the explanatory factors.

Finally, it can be mathematically formulated as:

$$r_i(x, \hat{\partial}) = \exp - \sum_{k=1}^{p} \partial_k (x_k - x_{jk})^2 \tag{3.31}$$

3.3 Neural Networks

Initially inspired by biology, neural networks have become increasingly popular, as they undoubtedly are a capable tool of machine learning. Much imitating the way the biological neural system operates (to the best of our current scientific understanding) they can be utilised to learn various association tasks and approximate complex non-linear functions. They are mostly separated into feed forward artificial neural networks (*ANNs*), and recurrent neural networks (*RNNs*). The first kind, is primarily about feed-forward networks capable of processing patterns without temporal association. Whereas the second one, which besides feed-forward connections also contain feedback loops to preserve, in the form of the information processing state, can handle temporal dependencies.

The complete complexity of the brains of all but the most primitive living things cannot be represented with artificial neural networks (ANNs). ANNs typically have a few hundred (or thousand) neurons at most, with extremely few connections between them. However, relatively tiny neural networks have been employed to address challenging computational issues.

3.3.1 Perceptron

The perceptron, first introduced by Frank Rosenblatt in 1957 is the most basic form of a neural network. Being a kind of artificial neuron, It is intended to simulate the human brain's basic decision-making process. Through a series of numerous binary inputs, weight multiplications, and results sums the necessary binary classification is made. Furthermore, in order to produce a singly binary output out of this whole process, a threshold function is applied. The perceptron's fundamental characteristic is its capacity to learn the aforementioned weights from the input-output pairs it is trained on, minimising the discrepancy between the desired and actual outputs by changing them using a straightforward learning procedure. While its ability to tackle non-linearly separable problems is limited, it is a fundamental idea in machine learning [24]. Below is presented the perceptron algorithm:

Algorithm 1: Simple Perceptron

Given:

$\mathbf{X}$ = input features with $\mathbf{x}_i \in \mathbb{R}^n$ for $i = 1, \ldots, m$

$\mathbf{y}$ = target outputs with $y_i \in \{-1, 1\}$

η = learning rate

Initialize $\mathbf{w} = \mathbf{0}$ and bias $b = 0$

Repeat until convergence:

For each $(\mathbf{x}_i, y_i)$ in the training set:

Calculate $\hat{y}_i = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i + b)$

If $y_i \cdot \hat{y}_i \leq 0$:

$\mathbf{w} = \mathbf{w} + \eta y_i \mathbf{x}_i$

$b = b + \eta y_i$

Output: final weight vector $\mathbf{w}$ and bias b

3.3.2 Multilayer Perceptron (MLP)

The most popular and widely used kind of neural network is the multilayer perceptron. Most of the time, signals travel through the network in one way only—from input to output. Each neuron's output has no effect on the neuron itself, so there is no loop. We refer to this architecture as feed-forward, and call hidden layers those that are not immediately visible to the outside world. The initial layer of the network, known as the input layer, is debatably regarded in the reference material as a stand-alone layer in and of itself because it serves the sole purpose of transmitting input signals to the upper strata without undergoing any input processing. We will only count the layers that are made up of standalone neurons, however we will note that the inputs are aggregated in the input layer. Additionally, there also exist the so-called feed-back networks. Because of response links within the network, they are able to transfer impulses in both directions. These networks have great power and have the potential to be highly complex. They are dynamic, constantly shifting until the network reaches an equilibrium state, and with every change in input, they hunt for a new balance.

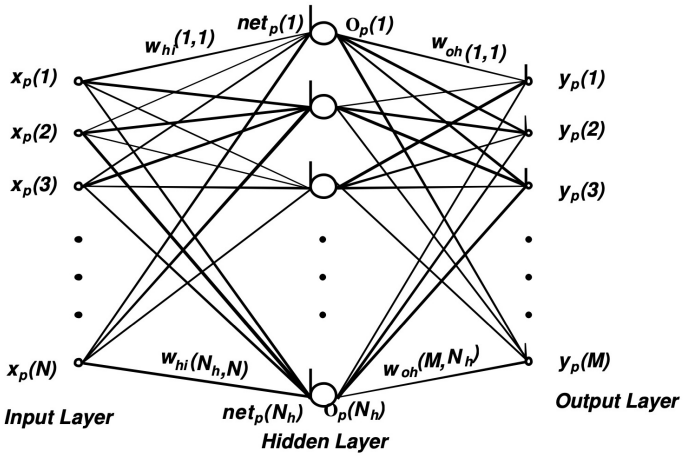


Figure 3.4

In the above figure, one can clearly see the distinction between each layer. The input and output layers are the ones communicating with the outside world. Generally, the more complex the problem, the bigger the number of neurons and layers needed in order to come up with an acceptable solution. In order for the whole system to work, a special kind of functions are needed, called non-linear activation functions. It is precisely these that give the multilayer perceptron its potency. Except for polynomial functions, almost any non-linear function can be applied in this way. One of the most commonly used is the single-pole sigmoid function, whose mathematical equation is displayed below:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.32}$$

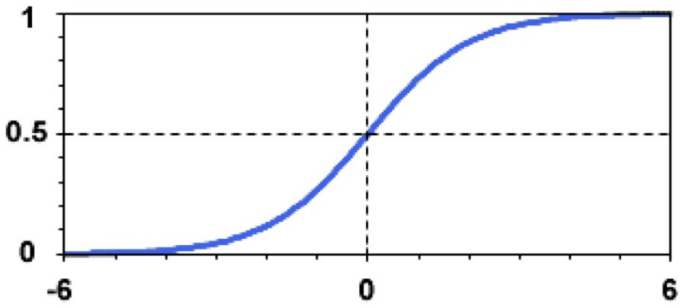


Figure 3.5: Sigmoid single-pole activation function.

The main logic behind the use of the single-pole activation function is the categorization of the output between two extreme values, 0 and 1. Depending on the problem setting, one can choose a variety of activation functions and even those extremes, mapping inputs to an output range that spans both positive and negative values, typically between -1 and 1. The last kind, comprises of the so-called bipolar functions, which are useful in scenarios where the model needs to distinguish between two opposite classes or directions more explicitly than just the presence or absence of a single class. The fact that this kind of function is able to represent negative outputs

directly, is particularly convenient in neural networks where neurons might need to represent negative activation. The bipolar sigmoid function is defined as:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1 \tag{3.33}$$

where $\sigma(x)$ is the standard sigmoid function, defined in 3.37.

Taking a more mathematical approach to the analysis of MLPs, the training dataset comprises of a set of N_v training patterns (x_p, t_p), with p representing the pattern order number. Going back to the example of 3.4, x_p is equivalent to the p_{th} training pattern's N-dimensional input vector, while y_p is equivalent to the M-dimensional output vector obtained by the trained network [25]. Assuming linear activations for the input and output units, the input to the i_{th} hidden unit will be conveyed by the expression:

$$net_p(i) = \sum_{k=1}^{N+1} w_{hi}(i,k)x_p(k) \tag{3.34}$$

and the p_{th} training pattern's output activation is normally indicated by:

$$O_p(i) = f(net_p(i)) \tag{3.35}$$

Finally, the i_th output for the p_th training pattern being expressed by

$$y_p(i) = \sum_{k=1}^{N+1} w_{oi}(i,k)x_p(k) + \sum_{j=1}^{N_h} w_{oh}(i,j)O_p(j) \tag{3.36}$$

Just like with any other machine learning model, neural networks have some pretty straightforward performance metrics. These can express how close to the training data are the model's predictions. To start, the mean square error (MSE), which can be stated as follows, measures the MLP's overall performance:

$$E = \frac{1}{N_v}\sum_{p=1}^{N_v} E_p = \frac{1}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}(t_p(i) - y_p(i))^2 \tag{3.37}$$

where E_p is the intended output for the p_th pattern, and t_p corresponds to the error for the p_th pattern.

3.3.3 Training of Multilayer Perceptron Neural Networks

Machine learning is usually accomplished in a supervised way. Presumably, a learning environment including both the learning models and models of the desired output matching to the input (referred to as "target models") is accessible. As we'll see, the standard basis for learning is the reduction of measurement errors between the desired outputs and the network outputs. This suggests that the learned network will propagate backwards across a comparable one. Thus, an especially widespread learning algorithm is called **back-propogation**. It was first launched in the mid-80s by Williams as a widely used instrument for multilayer perceptron training [25]. The idea is fairly simple: determining the smallest error function $e(w)$ with respect to the weights of the connections.

Step 1: First, let's initialise. First, random values are used to initialise the network weights and thresholds. These values are dispersed equally within a limited range, such as $(\frac{-2.4}{F_i}, \frac{2.4}{F_i})$, where Fi is the total number of inputs of the neuron i. In the event that these values are zero, the network will not learn because, in the absence of a direct link between input and output, the gradients that are computed during the trial will also be zero. In order to determine the optimal value for the cost function (least error), many training attempts with various initial weights are recommended. On the other hand, big beginning values have a tendency to saturate these units. The generated sigmoid function in this instance is quite tiny. It functions as a multiplication factor during learning, almost blocking the saturated units and causing learning to proceed very slowly.

Step 2: Presenting every example in the training set is referred to as an era. More training epochs are typically required for network training. The weights won't be changed until the network has seen the application of every test vector in order to preserve mathematical rigour. Therefore, after each model in the training set, the gradients of the weights must be remembered and adjusted. At the end of a training epoch, the weights will only be changed once (there is a simpler "on-line" variant in which the weights are updated directly; in this case, the order in which the network's vectors are presented may matter).The initial values of wij = 0 and E = 0 correspond to the gradients of the weights and the current error.

Step 3: Forward propagation of the signal

3.1 The inputs are applied with an example from the training set.

3.2 A calculation is made using the hidden layer neurons' outputs:

$$y_j(p) = f(\sum_{i=1}^{n} x_i(p)w_{ij} - \partial_j) \tag{3.38}$$

where f is the sigmoid activation function and n is the number of inputs for neuron j from the buried layer.

3.3 The network's actual outputs are computed as follows:

$$y_k(p) = f(\sum_{i=1}^{m} x_j k(p)w_{jk}(p) - \partial_k) \tag{3.39}$$

where m is the total number of inputs from the output layer that neuron k receives.

3.4 The error per epoch is updated:

$$E = E + \frac{(e_k(p))^2}{2} \tag{3.40}$$

with $e_k(p)$ equaling

$$e_k(p) = y_{d,k}(p) - y_k(p) \tag{3.41}$$

Step 4: Modifying the weights and propagating the errors backward.

4.1 The gradients of the mistakes are computed for the neurons in the output layer:

$$\delta_k(p) = f' e_k(p) \tag{3.42}$$

with f' being the derived function for the activation. The single-pole's activation function derived equals

$$\frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x)) \tag{3.43}$$

while the bipolar's is

$$f'(x) = \frac{2ae^{-ax}}{(1 + e^{-ax})^2} \tag{3.44}$$

4.2 The weight gradients between the output layer and hidden layer are updated:

$$\Delta w_{jk}(p) = \Delta w_{jk}(p) + y_j(p)\delta_k(p) \tag{3.45}$$

4.3 The gradients of the mistakes for the hidden layer neurons are computed:

$$\delta_j(p) = y_j(p)(1 - y_j(p)) \sum_{k=1}^{l} (\delta_k(p) w_{jk}(p) \tag{3.46}$$

where l represents the network's number of outputs.

4.4 Between the input layer and the hidden layer, the weight gradients take new values:

$$\Delta w_{ij}(p) = \Delta w_{ij}(p) + x_i(p)\delta_j(p) \tag{3.47}$$

Step 5: Another iteration.

The versatility and efficiency of this particular algorithm make it applicable across a vast array of problems, from image and speech recognition to natural language processing. It is fundamentally driving the success of modern AI applications.

Chapter 4

Constraint and Integer Programming

In order to properly define and examine various constraint programming techniques, it is necessary to first specify what a linear program actually is. When presented with extremely complicated practical problems, humans can now set broad goals and determine a course of specific decisions to follow in order to "best" attain those goals. This is where linear programming fits into a larger, revolutionary breakthrough. Our tools for doing this include engines for carrying out the stages of algorithms (computers and software), methods for solving the models (algorithms), and techniques to define real-world problems in comprehensive mathematical terms (models) [26].

4.1 Definitions

In a more formal expression, linear programming is a mathematical modeling technique in which a linear function is maximized or minimized when subjected to various constraints [27]. This method has proven helpful in industrial engineering, corporate planning, and, to a lesser degree, the social and physical sciences when making quantitative decisions. Getting to what actually will be covered in this section, according to Wolsey [28], "an integer program is a linear program where some or all decision variables are constrained to take on integer values only.". On the other hand, constraint programming (CP) is the name given to identifying feasible solutions out of a very large set of candidates, where the problem can be modeled in terms of arbitrary constraints. Due to their vast applications in engineering and problem solving, the two aforementioned techniques have been adopted by a number of object-oriented languages.

The goal of contemporary constraint-based languages is to make it easier to write readable, adaptable, and maintainable models. This is inherently difficult since complicated problem programs frequently call for the developer to use creativity to articulate several orthogonal issues and effectively encapsulate them within a language that has its own constraints. For good reason, logic programming is regarded as the foundation of constraint programming because it provides two crucial supports: non-determinism to enable search techniques, especially depth-first search,

and a declarative framework to define constraints as generalisations of unification. Nevertheless, there are several restrictions with logic programming. Initially, it is not conducive to the effective integration of expandable toolkits. Logic programming systems were first driven by simplicity and efficiency concerns to incorporate all constraints as "built-ins" into the language, which resulted in closed black-box solvers. Secondly, it is difficult to adapt search protocols that don't follow the depth-first approach [29].

4.2 Constraint - Based Tools

Tools for making combinatorial optimisation issues easier to design and implement have advanced significantly over the past 20 years. Their objective is to significantly reduce development time while maintaining the majority of specialised programs' effectiveness. The majority of tools fall into one of two groups: constraint programming languages or mathematical modelling languages. High-level algebraic and set notations are provided by mathematical modelling languages like AMPL and GAMS to represent mathematical problems succinctly, which can then be solved with the aid of cutting-edge solvers. These modelling languages are accessible to a broad audience and do not require specialised programming knowledge.

4.2.1 OPL

OPL is a modelling language that, like traditional modelling languages, shares high-level algebraic and set notations. Additionally, it has certain unique features to take use of sparsity in large-scale applications, like the capacity to index arrays using any kind of data structure. Rich constraint languages, support for scheduling and resource allocation issues, and the capacity to define search tactics and procedures are all features that OPL has in common with constraint programming languages. Additionally, OPL simplifies the process of integrating many solver technologies for a single application [30].

For instance, the goal of the frequency-allocation problem is to assign frequencies to many transmitters in a way that minimises the number of frequencies allotted and prevents transmitter interference. This problem is a real-world mobile phone problem in which the network is separated into cells, each of which has a number of transmitters with known positions. The following specifications apply to the interference constraints:

i) Within a cell, the separation between two transmitter frequencies cannot be less than 16.

ii) The separations between two transmitter frequencies from distinct cells are represented in a matrix and change based on their geographic location.

Naturally, the challenge is to assign frequencies to transmitters in a way that minimises the number of frequencies while also preventing interference. The remainder of this part is on utilising a heuristic to identify a solution that lowers the total number of frequencies allotted. In the following figures 4.1, 7.4, there will be shown an OPL statement for the frequency-allocation problem and the instance data respectively.

```
int nbCells = ...;
int nbFreqs = ...;
range Cells 1..nbCells;
range Freqs 1..nbFreqs;
int nbTrans[Cells] = ...;
int distance[Cells,Cells] = ...;

struct TransmitterType { Cells c; int t; };
{TransmitterType} Transmits = { <c,t> | c in Cells & t in 1..nbTrans[c] };
var Freqs freq[Transmits];

solve {
   forall(c in Cells & ordered t1, t2 in 1..nbTrans[c])
      abs(freq[<c,t1>] - freq[<c,t2>]) >= 16;

   forall(ordered c1, c2 in Cells : distance[c1,c2] > 0)
      forall(t1 in 1..nbTrans[c1] & t2 in 1..nbTrans[c2])
         abs(freq[<c1,t1>] - freq[<c2,t2>]) >= distance[c1,c2];
};

search {
   forall(t in Transmits ordered by increasing <dsize(freq[t]),nbTrans[t.c]>)
      tryall(f in Freqs ordered by decreasing nbOccur(f,freq))
         freq[t] = f;
};
```

Figure 4.1: Frequency Allocation Problem

The first interesting feature of the model is how variables are declared:

```
struct TransmitterType { Cells c; int t; };
{TransmitterType} Transmits = { <c,t> | c in Cells & t in 1..nbTrans[c] };
var Freqs freq[Transmits];
```

Figure 4.2

The problem statement makes it apparent that transmitters are housed inside cells. This structure is maintained by the aforementioned declarations, which is helpful when expressing limitations. All that is needed to define a transmitter is a record that has both a transmitter number and a cell number. The transmitter set is automatically calculated from the data used

```
{TransmitterType} Transmits = { <c,t> | c in Cells & t in 1..nbTrans[c] };
```

Figure 4.3

```
nbCells = 25;
nbFreqs = 256;
nbTrans = [8 6 6 1 4 4 8 8 8 8 4 9 8 4 4 10 8 9 8 4 5 4 8 1 1];
distance = [
   [16 1 1 0 0 0 0 0 1 1 1 1 1 2 2 1 1 0 0 0 2 2 1 1 1]
   [1 16 2 0 0 0 0 0 2 2 1 1 1 2 2 1 1 0 0 0 0 0 0 0 0]
   [1 2 16 0 0 0 0 0 2 2 1 1 1 2 2 1 1 0 0 0 0 0 0 0 0]
   [0 0 0 16 2 2 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1]
   [0 0 0 2 16 2 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1]
   [0 0 0 2 2 16 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1]
   [0 0 0 0 0 0 16 2 0 0 1 1 1 0 0 1 1 1 1 2 0 0 0 1 1]
   [0 0 0 0 0 0 2 16 0 0 1 1 1 0 0 1 1 1 1 2 0 0 0 1 1]
   [1 2 2 0 0 0 0 0 16 2 2 2 2 2 2 1 1 1 1 1 1 1 0 1 1]
   [1 2 2 0 0 0 0 0 2 16 2 2 2 2 2 1 1 1 1 1 1 1 0 1 1]
   [1 1 1 0 0 0 1 1 2 2 16 2 2 2 2 2 2 1 1 2 1 1 0 1 1]
   [1 1 1 0 0 0 1 1 2 2 2 16 2 2 2 2 2 1 1 2 1 1 0 1 1]
   [1 1 1 0 0 0 1 1 2 2 2 2 16 2 2 2 2 1 1 2 1 1 0 1 1]
   [2 2 2 0 0 0 0 0 2 2 2 2 2 16 2 1 1 1 1 1 1 1 1 1 1]
   [2 2 2 0 0 0 0 0 2 2 2 2 2 2 16 1 1 1 1 1 1 1 1 1 1]
   [1 1 1 0 0 0 1 1 1 1 2 2 2 1 1 16 2 2 2 1 2 2 1 2 2]
   [1 1 1 0 0 0 1 1 1 1 2 2 2 1 1 2 16 2 2 1 2 2 1 2 2]
   [0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 16 2 2 1 1 0 2 2]
   [0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 16 2 1 1 0 2 2]
   [0 0 0 1 1 1 2 2 1 1 2 2 2 1 1 1 1 2 2 16 1 1 0 1 1]
   [2 0 0 0 0 0 0 0 1 1 1 1 1 1 1 2 2 1 1 1 16 2 1 2 2]
   [2 0 0 0 0 0 0 0 1 1 1 1 1 1 1 2 2 1 1 1 2 16 1 2 2]
   [1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 1 16 1 1]
   [1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 2 2 1 16 2]
   [1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 2 2 1 2 16]];
};
```

Figure 4.4: Instance Data for the Frequency Allocation Problem

In this approach, there are two primary categories of restrictions. The distance restrictions between transmitters within a cell are handled by the first set of constraints. The command

```
forall(c in Cells & ordered t1, t2 in 1..nbTrans[c])
    abs(freq[<c,t1>] - freq[<c,t2>]) >= 16;
```

Figure 4.5

enforces the requirement that there be a minimum of 16 distance between two transmitters inside a cell. The main reasons the instruction is compact are that the forall statements allow us to quantify several variables, and the ordered keyword ensures that the statement takes into account triples <c,t1,2> where t1 < t2. The formulas freq[, t1>] and freq[, t2>] are particularly noteworthy as they demonstrate that the records of the type <c,t>, where c is a cell and t is a transmitter, are represented by the indices of the array freq. Also take note of the fact that the function abs, which determines the absolute value of its input (which might be any numeric expression),

is used to calculate the distance. The distance restrictions between transmitters from various cells are handled by the second set of constraints. The command

```
forall(ordered c1, c2 in Cells : distance[c1,c2] > 0)
    forall(t1 in 1..nbTrans[c1] & t2 in 1..nbTrans[c2])
       abs(freq[<c1,t1>] - freq[<c2,t2>]) >= distance[c1,c2];
```

Figure 4.6

argues that the distance between the frequencies of these transmitters must be at least the distance given in the matrix distance, taking into account each pair of different cells whose distance must be greater than zero and each two transmitters in these cells. The search technique of this model is another intriguing feature. The fundamental structure is not surprising: OPL selects a frequency in a nondeterministic manner, taking into account each transmitter. The heuristic is the model's intriguing characteristic. In the event of a tie, OPL selects the transmitter whose cell size is as tiny as feasible and produces a value for the transmitter with the smallest domain. A tuple (freq[t]),nbTrans [t.c]> is used to define this multi-criterion heuristic in order to obtain

```
forall(t in Transmits ordered by increasing <dsize(freq[t]),nbTrans[t.c]>)
```

Figure 4.7

4.2.2 MiniZinc

With its high degree of abstraction, MiniZinc may be used to formulate most CP issues in an easy and mostly solver-independent manner. It has support for user-defined predicates, sets, arrays, overloading, and certain automatic coercions. Nonetheless, MiniZinc is sufficiently low-level to be readily mapped onto a wide range of solvers. It is first-order, for instance, and it only accepts the decision variable types—integers, floats, Booleans, and sets of integers—that are supported by the majority of CP solvers currently in use. Some of the other features of MiniZinc are: it can separate a model from its data; it offers a library with declarative definitions of numerous global constraints; and it has an annotation system that lets declarative models be layered with non-declarative information (like search strategies) and solver-specific information (like variable representations) [31].

Solvers for constraint programming, such as Eclipse [32], Gecode [33], and Sicstus Prologue [34], currently support MiniZinc. The G12 FD solver, the hybrid G12 FD/SAT lazy clause generation solver [35], and the hybrid finite domain/linear programming solver are among the constraint programming solvers available in the G12 project that are capable of running MiniZinc. As MiniZinc is a solver-independent language, it can also accommodate different types of solvers. Think about the issue of reducing the highest possible amount of open stacks.

A factory produces a variety of goods in batches; that is, there are never two batches of the identical product because each copy of a given product must be completed

before another is produced. Every industrial customer submits an order for one or more distinct products. A stack is opened for each customer to keep all of the products in their order as soon as one of the products in their order begins to be created. An order can be submitted and the stack made available for use by another order once all of the products for a certain customer have been built. The goal is to ascertain which products should be produced in what order to reduce the greatest number of open stacks, or the largest number of customers with active orders at any given time.

For c customers and p products we have a $c \times p$ array orders of 0.1 which shows for each customer what products they order.

```
c = 5;
p = 9;
orders = [| 1, 0, 1, 0, 1, 0, 1, 0, 0
         | 1, 0, 0, 1, 0, 0, 1, 0, 0
         | 0, 1, 1, 0, 1, 1, 0, 1, 0
         | 0, 1, 0, 1, 0, 0, 0, 0, 1
         | 0, 0, 0, 0, 0, 1, 0, 0, 1
         |];
```

Figure 4.8: MiniZinc data file example.

```
% Open Stacks problem
include "all_different.mzn";
%-- Parameters ----------------------------------------------------%
int: c; % number of customers
int: p; % numer of products
set of int: Custs = 1..c;
set of int: Prods = 1..p;
array[Custs,Prods] of 0..1: orders; % products for each customer
array[Custs] of int: norders =
            [ sum(j in Prods)(orders[i,j]) | i in Custs ];
    %% total orders of each customer
%-- Decision variables -------------------------------------------%
array[Prods] of var Prods: s :: is_output ; % schedule of products
%-- Auxiliary variables ------------------------------------------%
array[Custs, 0..p] of var 0..p: o ; % # orders filled after time t
%-- Constraints --------------------------------------------------%
constraint all_different(s); % each product scheduled once
constraint
       forall (i in Custs)(o[i,0] = 0) % no orders at time 0
       /\
       forall(t in 1..p, i in Custs)(
            o[i,t] = o[i,t-1] + orders[i,s[t]] );
%-- Solving objective --------------------------------------------%
solve :: int_search(s, input_order, indomain, complete)
   minimize
      max(j in Prods)(
          sum(i in Custs)(
              bool2int( o[i,j-1] < norders[i] /\ o[i,j] > 0 )));
```

Figure 4.9: MiniZinc model file for open stacks.

The type of problem being solved is indicated by the item that begins with the keyword solve. Here, the objective function—which is the maximum of the open stacks at any given time—must be minimised. For customers who have at least one product built, but not all of them, it counts open stacks. The test's Boolean result is forced into a 0..1 integer using the *bool2int* function. There is a basic built-in search annotation on the solve item. The search is stored in annotations since the model is intended to be independent of the method used to solve it. This search annotation indicates that you should perform a thorough search and fix all the variables in the entered order, attempting their values from lowest to highest. The output of the above script is:

$$s = array1d(1..9, [1, 3, 5, 7, 2, 4, 6, 8, 9]); \tag{4.1}$$

Furthermore, to make decomposition building somewhat simple, MiniZinc offers reflection functions like as ub, lb, dom (which return the defined upper, lower bounds, and domain of variables or arrays of variables) and index set (which return the index set of an array). Here is an example of the sequence constraint's cumulative sum decomposition [36]:

```
%-----------------------------------------------------------------%
% Requires that in each subsequence
%          'x[i], ..., x[i + l - 1]'
% the sum of the variables is between 'mn' and 'mx'.
%-----------------------------------------------------------------%
predicate
  sequence(array[int] of var int: x, int: l, int: mn, int: mx) =
    let { int: iu = max(index_set(x)),
          int: il = min(index_set(x)),
          int: n = iu - il+1,
          int: maxs = n * ub(x),
          int: mins = n * lb(x),
          array[il-1..iu] of var mins..maxs: s } in
    s[i1-1] == 0 /\
    forall(i in il..iu)(s[i] == x[i] + s[i-1]) /\
    forall(i in il-1..iu-l)(
        mn <= s[i+l] - s[i] /\ s[i+l] - s[i] <= mx);
```

Figure 4.10

Beyond standard programming features, MiniZinc can allow more complex expressions as well. In order for this to be achieved, a process known as decomposition is used. For instance, the constraint

$$(x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2) \geq d; \tag{4.2}$$

with the variables ranging from 0 to 10, can be flattened to

```
1  var -10..10: t1 :: var_is_introduced :: is_defined_var;
2  var -10..10: t2 :: var_is_introduced :: is_defined_var;
3  var -100..100: t3 :: var_is_introduced :: is_defined_var;
4  var -100..100: t4 :: var_is_introduced :: is_defined_var;
5  constraint int_lin_eq([1,-1,1], [t1,x1,x2], 0) :: defines_var(t1);
6  constraint int_lin_eq([1,-1,1], [t2,y1,y2], 0) :: defines_var(t2);
7  constraint int_times(t1, t1, t3) :: defines_var(t3);
8  constraint int_times(t2, t2, t4) :: defines_var(t4);
9  constraint int_lin_ge([1,1,-1], [t3,t4,d], 0);
```

Figure 4.11

Common subexpression removal and (basic) limits analysis are performed during the flattening process to try to prevent the introduction of unbounded integer variables. When flattening nested expressions, MiniZinc takes care to maintain the relational semantics [37] of the model, which states that partial functions are deemed to result in the nearest enclosing Boolean context being false when the partial function's result is not declared.

4.3 Constraint Modeling

Two issues are brought up by constraint modelling: the toolkit's expressiveness and usability, as well as its inherent extensibility. Each issue directly affects possible end users and is inextricably related to the host language. Interesting issues arise when trying to model constraints in procedural or object-oriented languages. A declarative reading of a high-level model statement that makes use of the host language's capabilities—such as C++'s strong and static typing—is desired. Using the language to simplify programs and increase their modelling profile to a suitable degree of abstraction is challenging.

In terms of aggregation and combinators, think about the traditional magic series puzzle. The task at hand involves identifying a series of numbers $S = (s_0, s_1, ..., s_{n1})$ such that s_i denotes the number of times i appears in the sequence S. It is obvious that any solution needs to meet this requirement.

$$\sum_{k=0}^{n-1} (s_k = i) = s_i, i \in \{0, 1, 2, ..., n-1\} \tag{4.3}$$

It is first important to express the n constraints shown above in order to solve the problem with a constraint programming toolbox. Every constraint is a linear combination of elementary constraints of the form $s_k = i$ and their truth values, which are understood as 0 or 1. Building a toolbox with automatic constraint reification and seamless aggregation primitives—summations, products, conjunctions, and disjunctions, to mention a few—that makes combining basic primitives easier is consequently challenging.

The changes between the OPL and ILOG SOLVER statements for the magic series problem are shown in Figure 7.9. To build the cardinality constraint for every potential value, the ILOG SOLVER model iteratively produces an expression. It also

uses IloScalProd and other convenience methods to generate the linear redundancy constraint. Because the mathematical assertion maps directly to the model, the OPL model is relatively easier. It is important to note that, as illustrated in [38], C++ libraries can achieve the same level of abstraction as the OPL model while maintaining the same typing safety [29].

ILOG SOLVER

```
1. int main(int argc,char* argv[]){
2.   IloEnv env;int n;cin>>n;
3.   try {
4.     IloModel m(env);
5.     IloIntVarArray s(env,n,0,n);
6.     IloIntArray c(env,n);
7.     for(int i=0;i<n;i++) {
8.       IlcIntExp e = s[0] == i;
9.       for(int k=1;k<n;k++)
10.        e += s[k] == i;
11.      m.add(s[i] == e);
12.    }
13.    for(int i=0;i<n;i++) c[i]=i;
14.    m.add(IloScalProd(s,c) == n);
15.    solve(m,env,vars);
16.  } catch(IloException& ex) ...
17.}
```

OPL

```
1. int n<<"Number of Variables:";
2. range Dom 0..n-1;
3. var Range s[Dom];
4. solve {
5.   forall(i in Dom)
6.     s[i] = sum(j in Dom) (s[j]=i);
7.   sum(j in Dom) s[j]*j = n;
8. };
```

Figure 4.12: The Magic Series statements.

Respect for the host language's norms and values is necessary for a smooth toolkit assimilation. For example, C++ programmers frequently depend on the C++ compiler's type checking to detect errors and expect their programmes to have strong and static typing. Writing clear and straightforward models requires the ability to rely on types, particularly on finite domain variables defined over domains of particular kinds. Think about the issue of solid marriages. The challenge is to match men and women in a way that results in marriages and satisfies stability requirements based on everyone's choices. This is the OPL model in Figure 7.6. As can be seen from the fragment husband[wife[m]] = m, the values in the domain of wife[m] are of the enumerated type Women, which also happens to be the same type as the index for the array husband. In a similar vein, every entry in the husband array has the type Men, which makes it equal to the type of the equality constraint's right side. The outcome is a program that can be statically type verified, according to the modeller.

Writing brief and elegant models requires the ability to index arrays with finite domain variables, which is highly expressive. It is also helpful on matrices, particularly when its absence indicates that a tight reformulation based on an element constraint must be derived for an expression $m[x, y]$. A ternary relation is introduced by the reformulation.

$$R(i,j,k) = \{\langle i,j,k\rangle \mid i \in D(x) \wedge j \in D(y) \wedge k \in 0 \ldots |D(x)| \cdot |D(y)| - 1\}, i \in D(x), \quad j \in D(y) \tag{4.4}$$

It basically transfers the item m[i, j] to the array a's location k. Then, by including the constraint $(x, y, z) \in R$, where z is a new variable, $m[x, y]$ can be recast as $a[z]$. It should be noted that, as demonstrated in [38], templated libraries that provide both automatic reformulations and static/strong typing may be written for languages that have a comprehensive parametric type system (such as C++).

```
1.  enum Women ...;
2.  enum Men ...;
3.  int rankW[Women,Men] = ...
4.  int rankM[Men,Women] = ...
5.  var Women wife[Men];
6.  var Men husband[Women];
7.  solve {
8.    forall(m in Men)    husband[wife[m]] = m;
9.    forall(w in Women)  wife[husband[w]] = w;
10.   forall(m in Men & o in Women)
11.     rankM[m,o] < rankM[m,wife[m]] => rankW[o,husband[o]] < rankW[o,m];
12.   forall(w in Women & o in Men)
13.     rankW[w,o] < rankW[w,husband[w]] => rankM[o,wife[o]] < rankM[o,w];
14. }
```

Figure 4.13: The OPL model for Stable Marriage.

4.4 Relevant Paradigms

A basic application of complex constraint programming techniques are scheduling problems. In this subsection, we will conduct a thorough examination of similar cases to ours and solutions that were ultimately given. This way our optimization process in the end of this thesis will be largely facilitated. For instance, an extremely similar solution to our shift scheduling problem is the one regarding real time scheduling at junctions. The junction is probably going to be a vital resource when there is a lot of traffic on the connected lines. A well-known phenomenon states that tiny interruptions at junctions are amplified. Even a brief interruption at the source can cause a delay of more than five minutes. It is possible to construct new tracks, points, and fly-overs to lessen this occurrence. Investments and space are required for these extensions. These alternatives, however, are not likely to be practical in the near or medium future. In addition, they are typically impractical to execute in metropolitan settings. Therefore, in order to maximise the utilisation of limited resources, such as rail junctions, it is vital to look for innovative techniques and models [39]. Out of the steps that a traffic operator is following, the one that can most easily be changed is the selection and assessment of substitutes that cut down on conflict-related delays.

When under pressure, the operator might select subpar options. The aforementioned task can benefit from computer aid to enhance the quality of the final answer, as proposed by Fay and Schnieder (1997) [40]. However, in the end, the operator must make the final call. An optimisation module whose objective is to present a set of alternative options with the highest evaluation can be effectively incorporated into a decision support system. This module is triggered when an unforeseen circumstance, such a train delay, arises. The delayed train and any other trains that might be impacted by this unforeseen incident are included in the traffic's time horizon

that the module will be analysing. The operator must choose the horizon's length as one of the parameters. The difference between the time point at which the unforeseen occurrence is known and the time point at which the delayed train enters the junction determines the amount of time available for the module. Additionally, it must be considered that the operator requires enough time to evaluate the options offered and decide which one to apply. Having its goals extremely close to ours, the system aims at accuracy and short computation time.

Once again, a system serving as a simulator was used in order to model the complexity of the problem as accurately as possible. The behaviour of railway subsystems, including power supply networks, signalling systems, and train control, can be captured by a simulation model. Simulators generally employ an event-driven methodology. Discrete events are used to characterise the system's evolution. A simulation model may occasionally be incorporated into a control system as a predictive module, or for the purpose of anticipating future states, as demonstrated by Fernandez et al. (1994) [41]. The fundamental tenet of their constraint programming paradigm is that a train travelling through a junction represents a work. A job is defined as a collection of tasks connected by a set of precedence constraints, according to scheduling theory. A train moves through a series of actions. Every task involves the train moving elementally through a circuit of tracks.

We must introduce notation in order to define the problem in more formal terms. An sequence of n actions $A_1, ..., A_n$ is called a train run. Let $\mathcal{Z}$ be the set of m track circuit resources $Z_1, ..., Z_m$ for the infrastructure under consideration. The range of potential values for a variable is its domain in a CP formulation. Let $dom(x)$ represent an x variable's domain function. If r is a train route variable, then the train's alternate routes are denoted by $dom(r)$. $|R|$ is the number of track circuits of R for a route $R \in dom(r)$. The selection of a train's route becomes the allotment of resources (track circuits) for a series of tasks. Let $tc_1, ..., tc_n$ represent the resource variables that every activity needs. The set of track circuits $\mathcal{Z}$ is the starting domain of the variables tc_i. To make sure that our model is declarative, we construct a "fake" track circuit because not all possible routes have the same number of track circuits. Track circuit set Z is supplemented with the fictitious track circuit, designated Z^*. To obtain sequences of the same size for all possible routes, the fake track circuit Z^* is introduced into the track circuit sequence of a route. When a train has a route variable r, the length of its activity sequence is determined by

$$n = \max_{R \in \mathrm{dom}(r)} |R| \tag{4.5}$$

According to the article, assigning activities (train movements) to resources (tracks and signals) under a time constraint is the aim of a traditional scheduling problem, which is equivalent to the problem of managing train movements at junctions. The influence of the signalling system on the train movement scheduling is expressed by the CP model. The routing and scheduling problem is represented and resolved by the model using the following set of variables and constraints:

Train routes and their track circuit sequences are examples of variables. Various operational restrictions are included in constraints, such as track capacity (only one train on a track section at a time), scheduling restrictions (making sure trains adhere

to a workable schedule), and route dependencies between multiple trains to prevent collisions.

The model uses the idea of a "fake" track circuit to simplify calculation by standardising the sequence length across all routes, which allows it to manage routes with different lengths. This abstraction increases the applicability and flexibility of the model by enabling it to apply constraints consistently across various contexts. The CP model solution approach uses a search algorithm on a tree structure, with each node representing a possible solution that is determined by a set of variable assignments.

Firstly, in an effort to effectively narrow the search space, the model ranks variables according to how they affect the result. Then it tries removing unfeasible variable assignments to ensure consistency across constraints. This stage streamlines the solution process by utilising sophisticated techniques such as path-consistency and arc-consistency. Based on a heuristic that assesses the partial solutions, the model uses the Branch and bound technique which explores the most promising branches of the solution tree first in an effort to locate the best answer. Finally, via backtracking, the algorithm goes back to investigate alternate branches and makes adjustments to prior variable assignments in order to find a workable solution when a conflict is identified or a substandard branch is thoroughly examined. The approach is made to work with real-time restrictions, guaranteeing that solutions are identified quickly enough to be useful for on-the-spot train scheduling. In order to avoid cascading delays and operational disturbances, it is imperative that delays and timetable adjustments are swiftly managed in real-world railway operations.

All things considered, the model and the approach to solving it offer a solid framework for handling the difficulties of scheduling trains at junctions, fusing theoretical constraint programming methods with real-world demands for railway operations. The strategy is designed to take into account the highly regulated and dynamic nature of rail traffic, especially at crowded intersections, guaranteeing prompt and effective train transit through vital network nodes.

Chapter 5

Addressing The Problem

All models are wrong, but some are useful. (Box, 1979) [42]

Going back to the motivating illustration of section 2.1, it is evident that there is the necessity of training a model that will accurately and efficiently predict the simulation results of all the possible combinations of Motorcycles and Platforms (M, P). Furthermore, after identifying and successfully testing the particular model, we must make it operational in a realistic professional system. In this chapter, we will present our experimental findings that occurred by training and measuring the respective performance of over twenty machine learning models, most of which belong to the categories analyzed in chapter 2. After the analysis, we will present the apparently most efficient model in this prediction and the reasons why it adapts better to our problem. Finally, a fundamental problem to the solution will occur, which will be addressed in a later chapter of the thesis.

5.1 Issue Examination

In this section we will meticulously analyze the complexities of both the system and the problem at hand so that the reader gets a truly holistic view. This will facilitate how one can better understand the reasons why each particular model operated in the described respective manner. We have separated the system in 4 parts, each of which vital to the overall solution.

5.1.1 Intuitive Definition

As it will be evident in this chapter, this particular thesis delves into the optimization of both the Scheduling Solver 5.1.5 and the Surrogate Model 5.1.4. The whole problem can be considered to belong to both the resource allocation and the scheduling categories. Since the surrogate model predicts the incidents served and the average service time for all possible combinations of (M, P) in different times of the day and under different simulation conditions, one can say that it is a generalized resource allocation problem. This is because we try to identify the optimal combination of deployed vehicles for each hour, in order to achieve the company's goals. Furthermore, since all of this happens in the context of scheduling drivers' shifts according to certain rules, we can also conclude that the latter part of the problem is a scheduling

one.

But the truth is that most of the work of this thesis has been devoted to the surrogate model and the ability to ensure that its predictions work reliably in the context of a complete industrial system. This includes eliminating any edge cases and performing parameter fine-tuning of the model. The company has a need of optimizing the current model and eliminating the cases which force it to miss its service goals. This includes finding a new model and maybe even a new method of training it, and make its solutions adapt to the current system. This will be addressed in chapters 5 and 6, while in chapter 7 we will tackle the scheduling side of the issue.

5.1.2 Input Data

Input of the whole problem is the historical data of incidents on specific days. This includes information such as *querydate* (date and time of incident), location (coordinates), malfunction (in order to determine the type of vehicle needed), company (the company to which the incident belongs), service time (time the company's vehicle spent in the incident). As stated before, the types of possible available vehicles are two: "Motorcycles" and "Platforms". The main separator between the two is that the latter has the ability of towing. Additionally, the system accepts specific objectives in the form of Key Performance Indices (KPIs). These are also in dictionary format, with different information for each company such as:

PercentageUnderX	Percentage of accidents served in less than X minutes.
PercentageOverY	Percentage of accidents served in more than Y minutes.
AverageServiceTime	Average service time of all company incidents.
GoalMinutesX	Value of X
GoalMinutesY	Value of Y

Table 5.1: Main Key Performance Indices (KPIs)

Service time is defined as the time between the incident call and the moment of arrival of roadside assistance. For instance, a specific company may have set goals that $PercentageOverX = 80\%$ of its cases should be served in a time shorter than $GoalMinutesX = 45'$, and correspondingly that the maximum percentage of $PercentageOverY = 3\%$ should be served in a time longer than $GoalMinutesY = 180'$. This imposes a very specific boundary on the performance metrics the chosen model must have, since the model's capabilities play a vital role on whether or not the company's goals will be met. In this thesis, we will not address each company's goals, but we will rather form a general solution that covers one set of goals.

5.1.3 Simulator

The simulator mimics roadside assistance incident routing as it happens concurrently, with the ultimate goal of producing data for the Surrogate Model 5.1.4. In

practice, it is possible to use the simulator to discover the correlation between the number of resources and the objectives achieved. Its input is the historical incident data of a time period and combinations of available numbers of platforms and motorcycles produced by the Driver Availability Generator through uniform grid search sampling. For example, if the Driver Availability Generator produces the pair (M, P) = (6, 9) as available for the time 06:00-07:00, the aim of the simulator is based on the historical data to make virtual assignments and calculate the values of the KPIs at the specific time. So finally, the triad (M, P, KPI) is produced for the time 06:00 - 07:00. The triad programmatically consists of integers M, P, and the dictionary KPI which has the values of the corresponding percentages. It should also be noted that the way the Simulator assigns incidents is considered optimal and is not subject to improvement.

The Driver Availability Generator is a system that takes as input the different appointed driver shifts and calculates how many and with what kind of vehicle would be available at a particular time slot. Systems like this one and the simulator are arbitrarily considered to be working ideally, and their complexitites are out of context for the current thesis. Their role is important, yet complementary to the overall solution, which largely depends on the performance and effectiveness of the surrogate model. The latter, which is the focus of the first three chapters, is explained below.

5.1.4 Surrogate Model

Crucial to the needs of the problem is the accurate estimation of as many triads as possible (M, P, KPI) as explained in subsection 5.1.3. There is a need for research, appropriate selection and training of machine learning models which will provide in a relatively short period of time an accurate estimation of the aforementioned KPIs. Essentially, due to the limitations arising from the time needed to run the simulator, the model will ultimately have to accept specific triads and predict with relative accuracy the KPIs for all the remaining combinations. Visually, there is a function that has an input that optimizes its values on the KPIs axes while simultaneously minimizing wasted resources, i.e. platforms and machines. The aim of the model is by receiving some points of the function (curve) to predict the rest. Furthermore, the model must run for specific KPIs each time and for each company. That said, different runs are needed for PercentageUnderX and for PercentageOverY respectively. It is worth noting that such forecasts are needed for every hour of every calendar day, in order to later select routes and drivers for that specific time slot. The reason that forecasts are made separately for each hour is to isolate the relationship between resources and KPIs without being affected by the spatial and temporal distribution of incidents.

5.1.5 Scheduling Solver

The Integer Programming Solver is already implemented in python using OR Tools and more specifically the Element Constraint which expects a list of triples of all possible combinations (P, M) based on the available resources and the corresponding KPIs. As additional constraints, it has limitations of a business nature such as that each driver must work exactly 8 hours. For example, if the shifts start at 00:00,

there cannot be three platforms at 00:00 - 01:00 and two at 01:00 - 02:00. Based on the triads given by the Surrogate Model and the corresponding constraints, he must find the assignment that serves the most objectives with the least resources. Then, for each hour, it returns the trio that it "decided" to be the most ideal. Based on the results produced by the Scheduling Solver, the actual shifts of each driver must be derived. If for example for the day 30/11/2023 the Solver has produced the following table (normally this table will cover the whole twenty-four hours):

Time of Day	**Platforms**	**Motorcycles**	**KPIs**
13:00 - 14:00	14	29	[dictionary1]
14:00 - 15:00	12	31	[dictionary2]
15:00 - 16:00	12	33	[dictionary3]
16:00 - 17:00	14	27	[dictionary4]

Table 5.2: Solver Normal Output

The specific algorithm must decide who will work and when, respecting specific business constraints and following the table as closely as possible. The desired output of the entire program is the drivers' shift schedule that will result from the last stage. This process needs to be repeated several times. For example, based on the result of the first time and the resulting KPIs, the input of the second time will be determined and problems such as wrongly serving set goals or optimizing resource allocation will be solved.

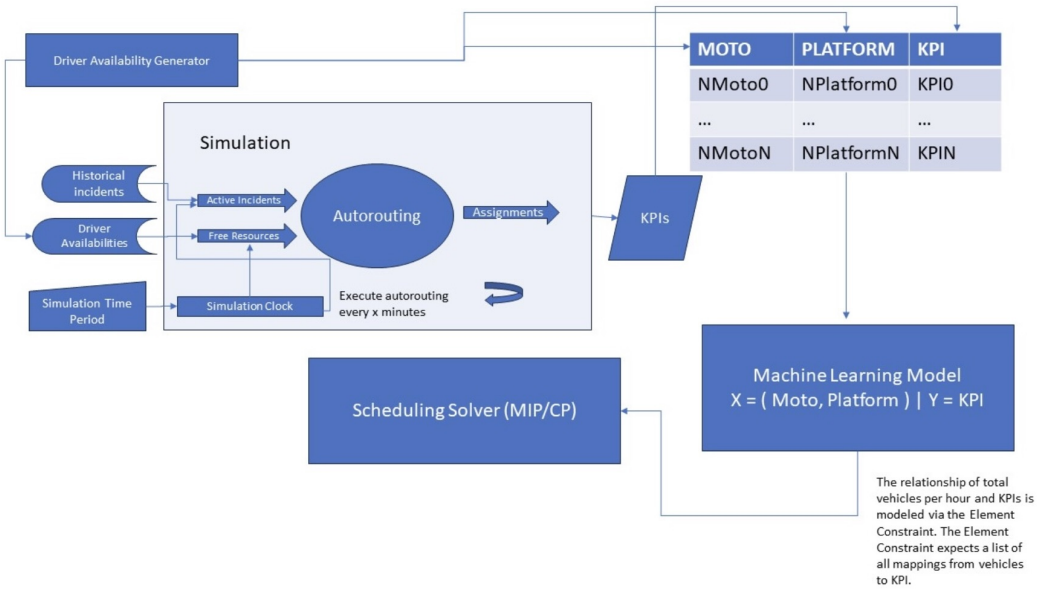


Figure 5.1: System's operational diagram

5.2 Surrogate Model Testing

In order to address properly the above problem and find the best fitting surrogate model, a different number of machine learning models were tested and evaluated by the standardized performance metrics discussed in chapter 2. Obviously, only a

small fraction of those tested proved to be suitable for our problem but in the name of completeness in this subsection we will mention most of them. Each model was trained on the 80% of data and then its adaptability was validated on the rest 20%. All the model training was conducted in python, utilizing the *scikit – learn* library and its capabilities.

5.2.1 Linear Models

Intuitively, one can instantly understand that the nature of our problem's solution is anything but linear. Besides the complex correlations between the input and output parameters, the mere size of the input vector X is a limiting factor for the Linear Models. Logically, only after numerous simplifications would the models make accurate enough predictions to be considered acceptable, but then the nature of the problem would be altered. Nonetheless, we tested eight linear machine learning models and evaluated their accuracy. They all are based on the multiple linear regression technique, each one with different modifications and enhancements. Not all of the linear models tested will be presented below, because of the overlapping nature of their results.

The obvious model that was tested first was the classic Linear Regression. It follows the mathematical equation

$$Y = X\beta + \epsilon \tag{5.1}$$

where ϵ is an n-vector of i.i.d. random variables (often considered Gaussian) with mean 0 and variance σ^2, X is a n $\times$ p matrix with individual unique covariate vectors as its rows, and Y is an n-vector of independently distributed answers. In our example, Y takes two seperate values, the number of incidents served (RoutedCount) and the average service time of each incident (AvgServiceTime). The model is trained twice, once for each variable. It uses the Ordinary Least Squares method (OLS) which basically aims to minimize the Residual Sum of Squares 3.14. After the training, the predictions were graphically:

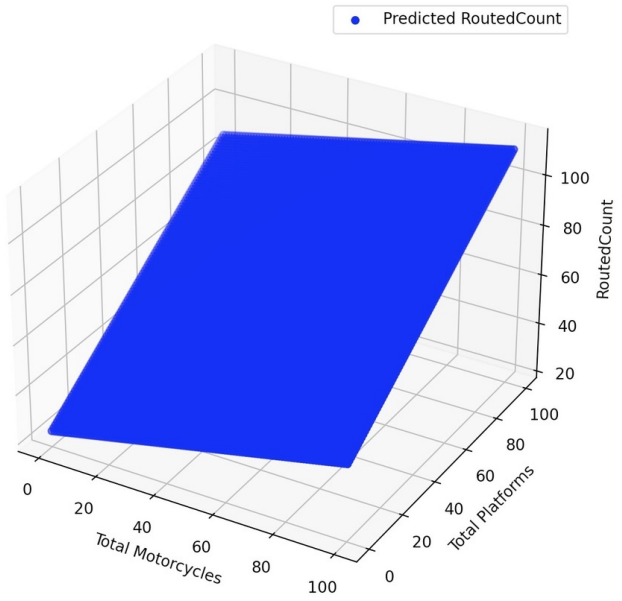


Figure 5.2: Linear Regression prediction for incidents served (RoutedCount).

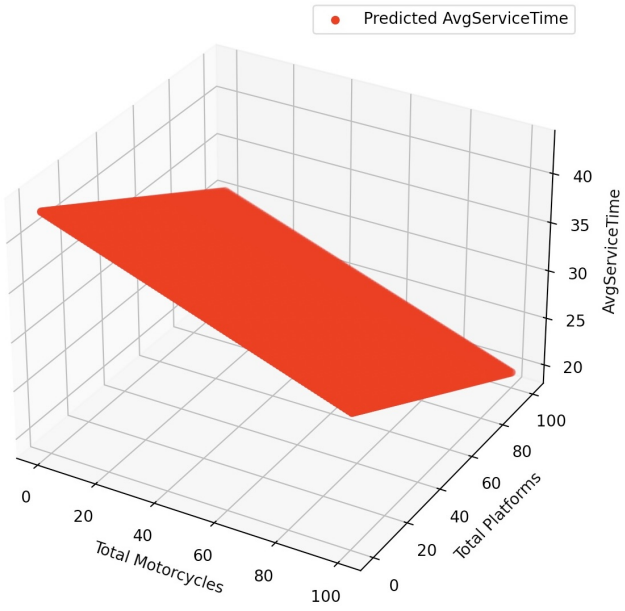


Figure 5.3: Linear Regression prediction for average service time of incidents.

As one can see, the prediction is fairly logical, with the number of incidents served being minimized (=0) when $(M, P) = (0, 0)$ and following a linear increasing course all the way to its maximisation at $(M, P) = (100, 100)$. A similar structure is followed for the average service time prediction, where its minimised for the maximum number of resources available and maximised for the minimum.

Even though a logical path is being followed, the evaluation metrics are unacceptably low, at

$$R^2 Score = 0.3791 \tag{5.2}$$

$$MSE = 362.4086 \tag{5.3}$$

for the RoutedCount prediction, and

$$R^2 Score = 0.1613 \tag{5.4}$$

$$MSE = 73.6881 \tag{5.5}$$

for the Average Service Time.

Another model tested was the Lasso regressor. Originally proposed by Tibshirani in 1996 [43], it takes advantage of the linear regression theory which as stated above, uses the equation 6.4. Standard least squares methods cannot uniquely estimate the unknown coefficient vector β when p > n. In reality, since standard errors are likely to be high and parameter estimations unstable, it is probably not a good idea to use least squares to estimate the vector even when n ≥ p and p close to n. To pick and estimate the nonzero parts of β in this case, assuming that β is relatively sparse with numerous zero entries, one can effectively use the Lasso estimator calculated by minimising

$$\frac{1}{2} ||Y - X\beta||_2^2 + \lambda ||\beta||_1 \tag{5.6}$$

where the level of regularisation is controlled by the predefined value λ.

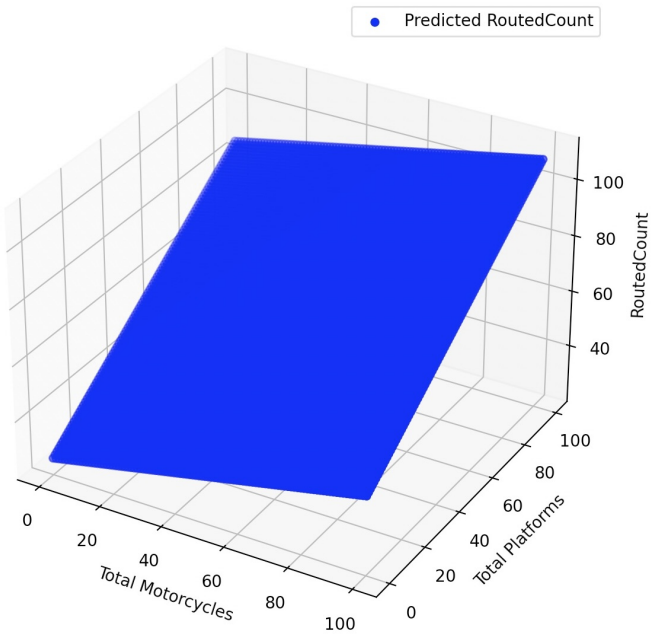


Figure 5.4: Lasso prediction for incidents served (RoutedCount), with $\lambda = 10$.

More elements of the estimated β vector are set to 0 and the nonzero entries are squeezed towards zero as λ increases. Less regularisation and more nonzero β with bigger (absolute) coefficients are implied by smaller λ. Therefore, by adjusting Y to be the number of incidents served each time and X a 2-dimensional vector, containing only the pair (M, P), our expectations are not particularly high. The results are presented for academic purposes.

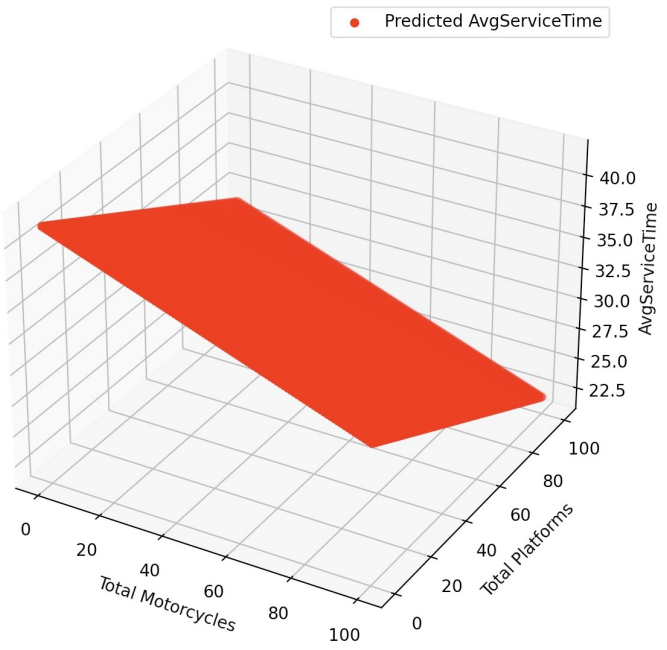


Figure 5.5: Lasso prediction for average service time of incidents, with $\lambda = 10$.

It is fairly obvious that both the RoutedCount and AvgServiceTime graphs produced by the Lasso regressor are almost identical to the ones produced by the classic linear regressor. Although $\lambda = 10$, which increases the regularization strength and can lead to sparser models (with more coefficients set to zero), the prediction is still far from acceptable.

Now, the respective performance metrics are slightly improved, mainly due to Lasso's higher adaptability compared to the traditional linear regressor:

$$R^2 Score = 0.4129 \tag{5.7}$$

$$MSE = 338.8964 \tag{5.8}$$

for the RoutedCount prediction, and

$$R^2 Score = 0.1672 \tag{5.9}$$

$$MSE = 69.1549 \tag{5.10}$$

for the Average Service Time. It remains so low because, on the one hand the nature of the solution of the problem is inherently non-linear and on the other, the Lasso regressor works best when the predictor vector is way bigger than the predicted variables [44].

In the name of finding a linear model that could produce a better performance, numerous were tested but none managed to improve the aforementioned ratings. Models that performed as well as the Lasso included LARS, Huber, ElasticNet, BayesianRidge and a few variations of them. All produced almost identical plots to 5.4 and 5.5, while the performance metrics were practically the same as the Lasso's. So, we deemed it unnecessary to present any more theories and results from any other linear models since it does not contribute further neither practically nor scientifically to the solution of the problem.

5.2.2 Non-Linear Models

Empirically, the solution to our problem seems to be fitting for a non-linear machine learning model. For the number of incidents served for example, its relationship with the pair (M,P) is definetely not linear since the platforms can serve all kinds of road incidents while the motorcycles only the ones that do not need any physical transportation of people or vehicles. On the other hand, motorcycles are way faster when serving an incident or being transported from one incident to the next. Such peculiarities influence the solution in ways that the human mind struggles to grasp, but non-linear models can recognise seemingly undetectable patterns that help them formulate a logical solution. Similar to the linear ones, many models were tested only this time with higher performance metrics.

One of the most famous non-linear models, is the polynomial regressor 3.2.1. The experimental results of the polynomial regressor of degree 2, are shown below:

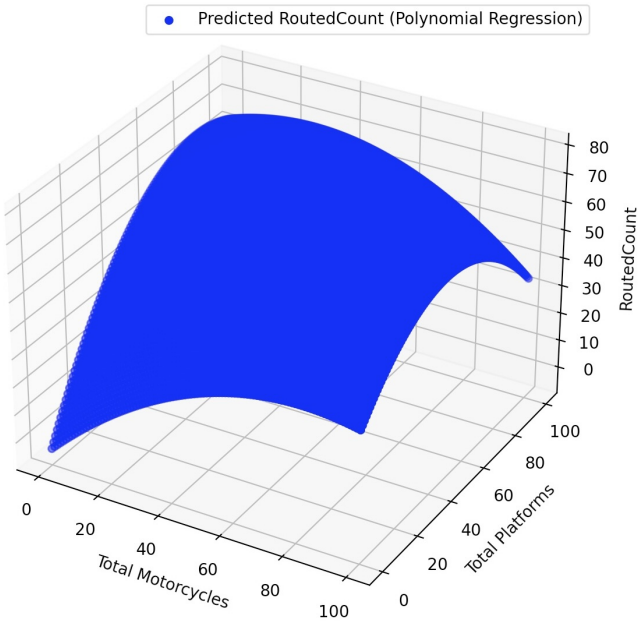


Figure 5.6: Polynomial of degree 2 prediction for incidents served (RoutedCount).

Graphically, the results do not seem as logical as the linear ones. Even though as we will see below the performance metrics have improved, the prediction implies that for the maximum number of vehicles the RoutedCount is not maximised. This, as we will thoroughly examine in a later section happens due to the fact that there are numerous factors besides the pair (M,P) that interact with the final number of incidents served or with the average service time. Initially, we adhere to this specific approach as it is pre-existing in the system, and our initial efforts are focused on enhancing it without making any substantial alterations.

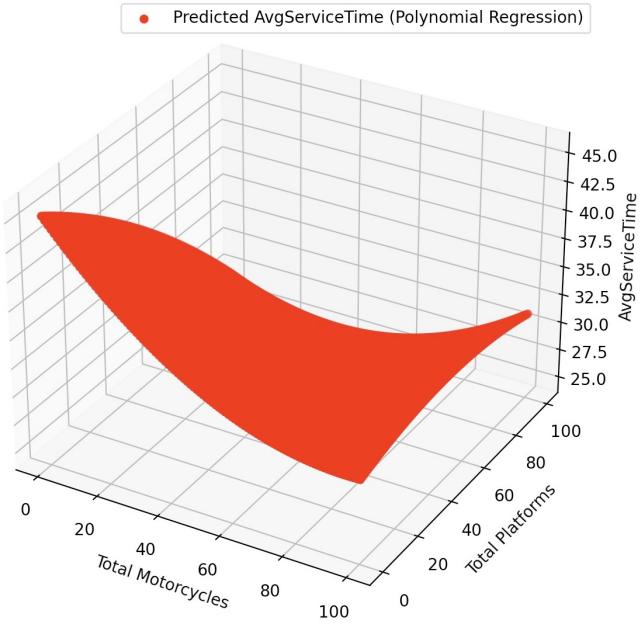


Figure 5.7: Polynomial of degree 2 prediction for average service time of incidents.

Analogous to the RoutedCount, the AvgServiceTime variable is not minimized when the number of vehicles are maximized. For this model, the performance metrics are improved compared to previous ones:

$$R^2 Score = 0.5192 \tag{5.11}$$

$$MSE = 277.5278 \tag{5.12}$$

for the RoutedCount prediction, and

$$R^2 Score = 0.1884 \tag{5.13}$$

$$MSE = 67.3920 \tag{5.14}$$

for the AvgServiceTime. Due to its curved nature, by adjusting its coefficients a second degree polynomial fits the solution noticeably better than its linear counterparts [17]. While improved, the fit to the data is still not acceptable and thus the model cannot be used for reliable predictions.

Next, the K-Nearest Neighbors (KNN) model 3.2.2 was tested. With its incredibly more adaptive nature, it formulates its prediction exclusively according to the training data it is presented to, without having any biased equation. This time, the performance metrics were significantly improved, showing us that this is the first model making an acceptable prediction. It generally performs well for datasets where similar instances lead to similar outcomes. Graphically, its results are not clear since they are based on the dataset itself. It should further be noted that the KNN model is the one currently being used to make the necessary predictions. Below lie our findings.

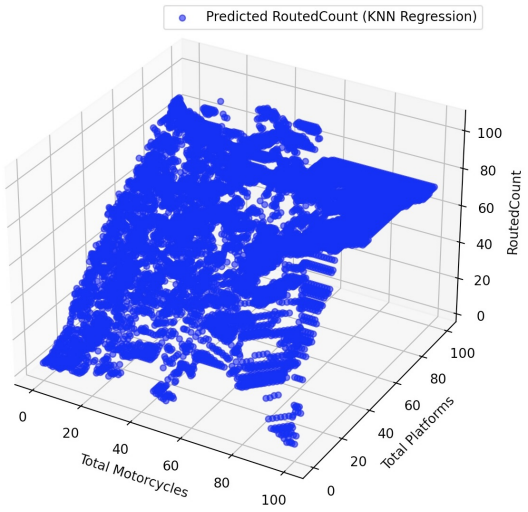


Figure 5.8: KNN model prediction with $K = 5$ for number of incidents served.

As one can easily notice, this model makes a clear distinction between the effect that the number of platforms and the number of motorcycles has on the total RoutedCount. Obviously, beyond a certain threshold, when the number of platforms is zero, any additional motorcycles do not impact the RoutedCount.

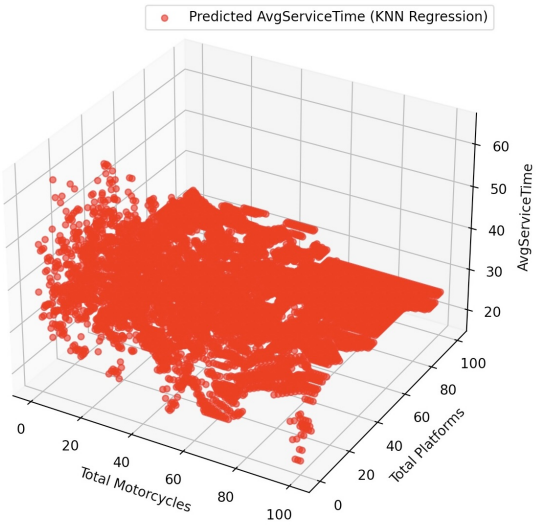


Figure 5.9: KNN model prediction with $K = 5$ for average service time of incidents.

The average service time prediction is a bit less accurate according to the respective metrics but still much improved. It follows a logical path in the three-dimensional plane, minimizing the service time when (M, P) are maximised. The performance metrics are displayed below.

$$R^2 Score = 0.8144 \tag{5.15}$$

$$MSE = 107.1085 \tag{5.16}$$

for the RoutedCount prediction, and

$$R^2 Score = 0.5661 \tag{5.17}$$

$$MSE = 36.0298 \tag{5.18}$$

for the AvgServiceTime. It is evident that the R^2 score for the service time is systematically lower than the one for the served incidents. This will be examined in a later section, and is mainly because the service time is affected by a number of parameters that are not being considered under the current model training system.

The question now becomes, whether there is a model capable of improving further the KNN's performance with the current training system. Moreover, the new model should have the potential of improving further when additional parameters are appended. So, with the above in mind, the next model that was tested is the Decision Tree regressor. By splitting the complex problem into simpler decisions, we hope that the current model will better adapt to the peculiarities of the issue and further improve the performance, especially regarding the AvgServiceTime variable [20]. The experimental results visually were:

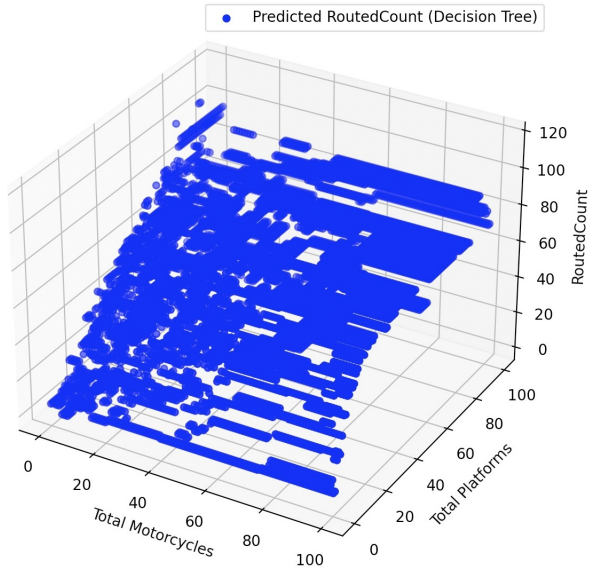


Figure 5.10: Decision tree model prediction for number of incidents served.

At first, one might notice a similar behaviour to the KNN model but this is not exactly the case. The prediction here is way more accurate, mainly due to the automatic feature selection conducted by the decision tree algorithm, giving higher importance to the most informative features for making splits. So, since the data is split based on feature values, decision trees adapt better to data where the relationship between features and the target variable is not straightforward. This applies extremely well to our specific dataset and system settings, since the relation between the pair (M, P) and the RoutedCount, AvgServiceTime values hides many underlying assumptions. With that being stated, here is the equally improved service time prediction graph:

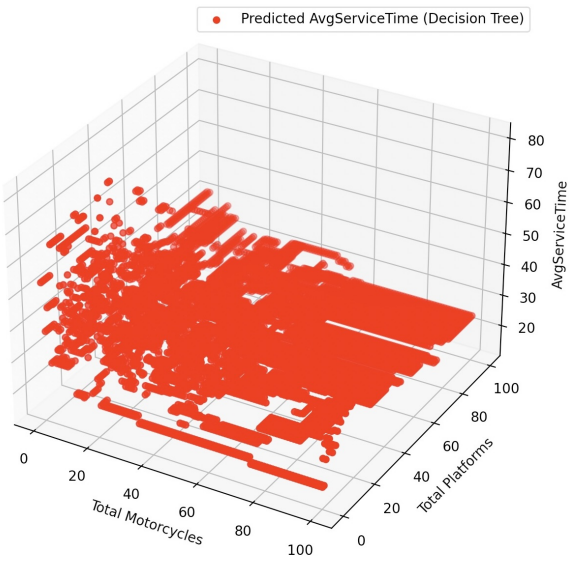


Figure 5.11: Decision tree model prediction for average service time of incidents.

This is the first service time prediction which can be considered acceptable, and a demonstration of the decision tree's ability to model complex and hierarchical relationships in the data, perform inherently feature selection, and efficiently capture feature interactions. The performance metrics speak for themselves.

$$R^2 Score = 0.8787 \tag{5.19}$$

$$MSE = 70.0104 \tag{5.20}$$

for the RoutedCount prediction, and

$$R^2 Score = 0.7108 \tag{5.21}$$

$$MSE = 24.0125 \tag{5.22}$$

for the AvgServiceTime.

As the performance metrics values increase, we should state that overfitting is not a current concern as the models will be tested with new data throughout the training process. A more detailed overview of the above will be presented in chapter 5 of the thesis.

Naturally following the decision tree regression algorithm is the Random Forest regressor. The theory behind it was meticulously analyzed in subsection 3.2.4. To increase prediction accuracy and reduce overfitting, it essentially builds several decision trees during training and outputs their average forecast. In order to create a more reliable and comprehensive model than a single decision tree could provide, it integrates the findings from many decision trees that are constructed using distinct subsets of the data and characteristics. By following the aforementioned principles this algorithm appears to be a unique solution to our problem, providing one of the most convincing predictions. Below lie the respective experimental findings.

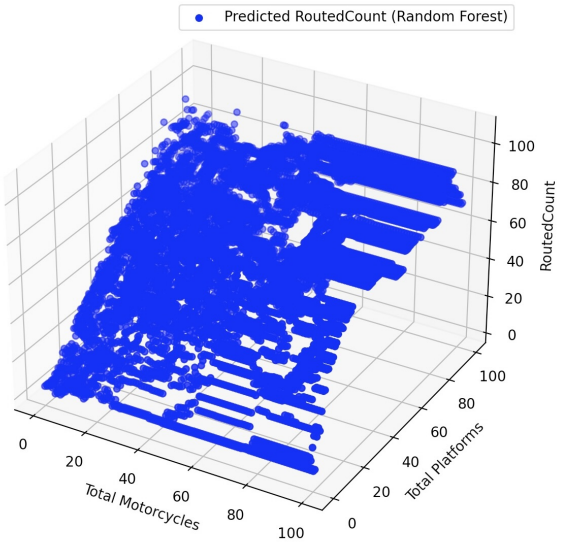


Figure 5.12: Random forest model prediction for number of incidents served.

This is a fairly similar prediction to the one made by the decision tree algorithm. The difference is that now a more holistic approach is taken to solving the problem, giving us the ability to slightly optimize the overall model's performance.

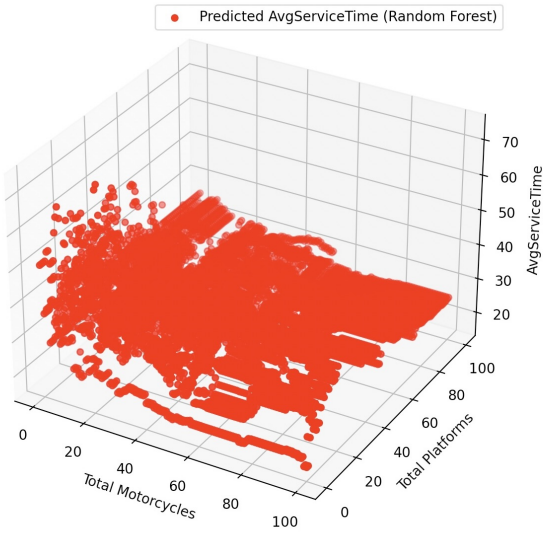


Figure 5.13: Random forest model prediction for average service time of incidents.

The performance metrics lie below.

$$R^2 Score = 0.8821 \tag{5.23}$$

$$MSE = 68.8390 \tag{5.24}$$

for the RoutedCount prediction, and

$$R^2 Score = 0.7190 \tag{5.25}$$

$$MSE = 23.5620 \tag{5.26}$$

for the AvgServiceTime.

The last model and its performance was the closest we could get to the ideal model fit and error minimization. A number of other models were tested but none topped the above prediction's accuracy. Since most have already been analysed in 2, and their contribution does not add anything to the solution of the problem, they will not be further examined. The most promising ones were Gradient Boosting 3.2.5 and 3.2.6 Gaussian Process, whose scores and predictions were analogous to the KNN's.

5.2.3 Multi-Layer Perceptron

The Multi-Layer perceptron's case is a rather interesting one. Usually, neural networks are used in order to solve problems more complex than their non-linear counterparts. While being computationally more expensive, their potential in terms of prediction accuracy in particularly difficult tasks is unmatched and this is the reason why they are referred to as the last resort in such problems. Because of their network-like nature, it is evident that while as amount of information given to them increases so will their accuracy in terms of data relevance [45]. But there is a catch. Along with the prediction's accuracy, the danger of overfitting and computational complexity also increase.

Based on the above and the analysis conducted in section 3.3, we approach the following test with caution and low expectations. These models are trained only based on two parameters which are the pairs of (M,P). While this is not the ideal case for the *MLP*, we mention its results because of its potential improvement when a different approach is followed.

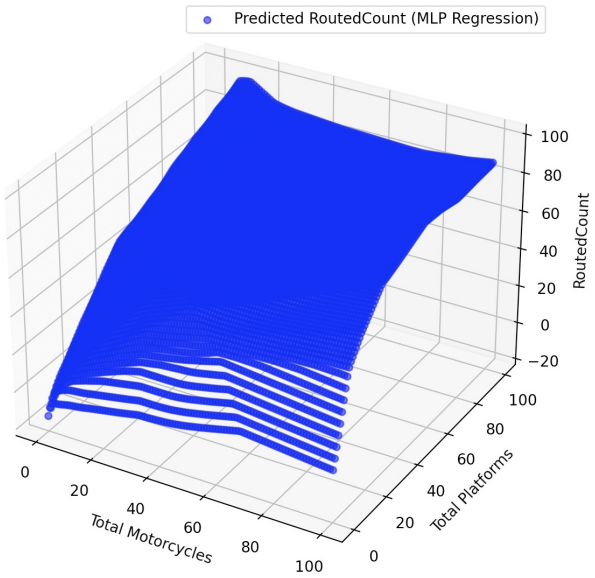


Figure 5.14: Multi-Layer Perceptron (MLP) prediction for number of incidents served.

Being a fundamentally different to the rest, its prediction pattern is also divergent. One can notice that the important weight to the total number of platforms given

by the last models, does not exist in this prediction. This actually is close to the polynomial model, which did not make a distinction between the Platforms and the Motorcycles in terms of affecting the prediction variables.

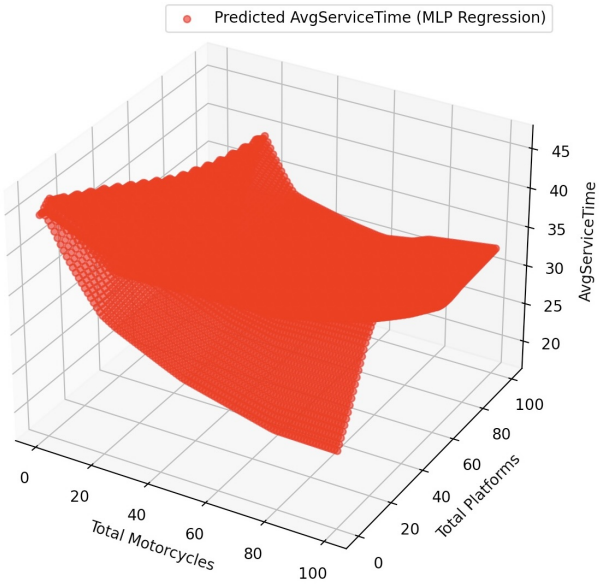


Figure 5.15: Multi-Layer Perceptron (MLP) prediction for average service time of incidents.

Logically, the MLP's performance metrics with the current approach will be extremely low compared to some of its aforementioned non-linear counterparts.

$$R^2 Score = 0.5753 \tag{5.27}$$

$$MSE = 245.1577 \tag{5.28}$$

for the RoutedCount prediction, and

$$R^2 Score = 0.2479 \tag{5.29}$$

$$MSE = 62.450 \tag{5.30}$$

for the AvgServiceTime. Once more, we should emphasize how important is the problem complexity for the MLP's performance. As we will discuss in a later section the model could be ideal in solving our problem, provided a different setting.

5.3 Edge Cases

As it is obvious from the previous section, our work has produced a reliable approximation of the ideal prediction having improved the previous performance threshold. The question now becomes whether or not the accuracy can be further enhanced. In the preexisting solution of the problem, there was a sizeable number of edge cases that were damaging the overall prediction of the algorithm. This next step is about identifying these edge cases and exploring ways that they can be diminished.

Firstly, for every model trained we created a histogram indicating the differences between the prediction values and the actual data from the testing set. The goal is to minimize the number of cases larger than a certain value, depending on the output variable predicted. For instance Lasso, a linear model with high *MSE* and low R^2 scores will have histograms:

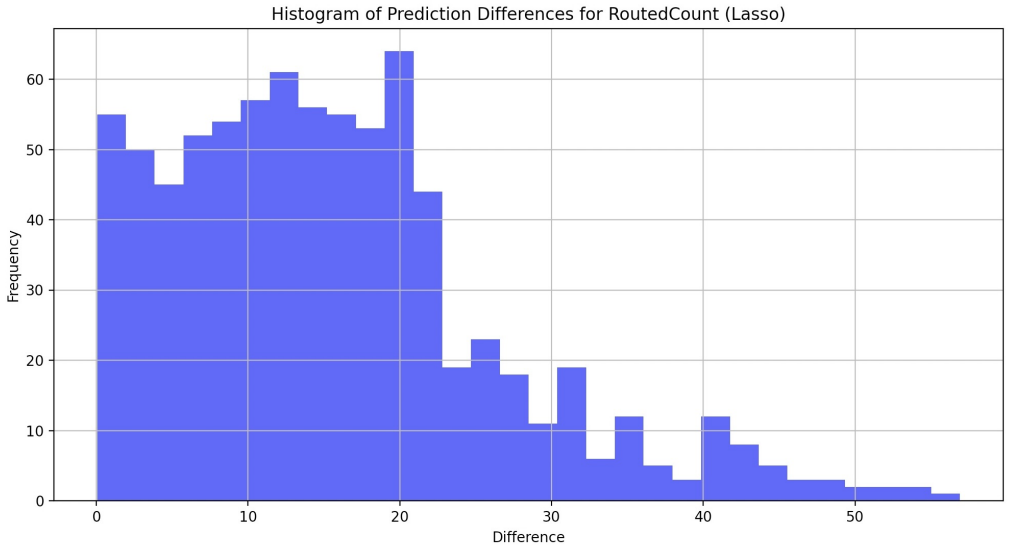


Figure 5.16

It is fairly easy to notice that the differences are unacceptably high for both the RoutedCount and AvgServiceTime variables.

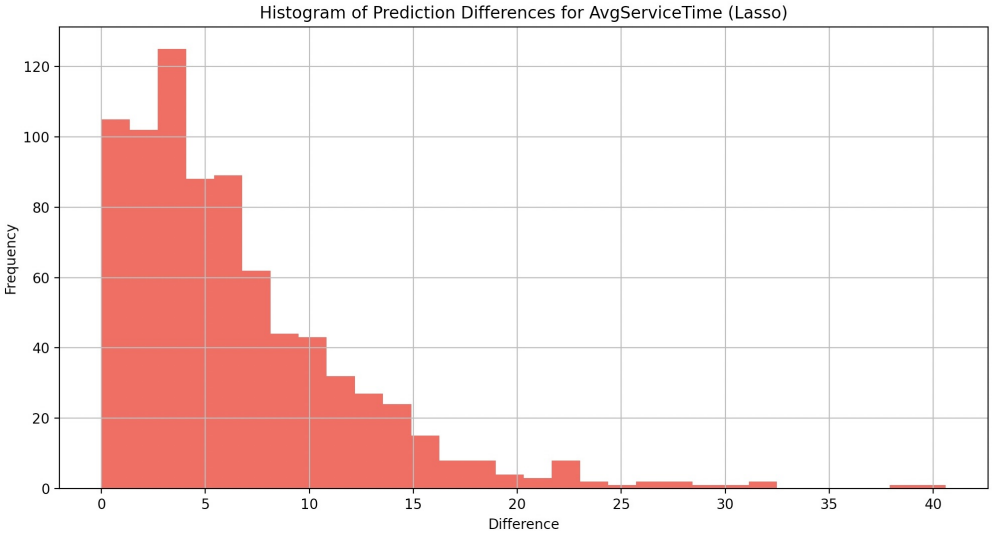


Figure 5.17

This issue is mitigated when transitioning to the most effective model, the Random Forest. Given its superior R2 score, this indicates a closer fit of the model to the data, thereby reducing the unexplained variance around the mean of the response variable.

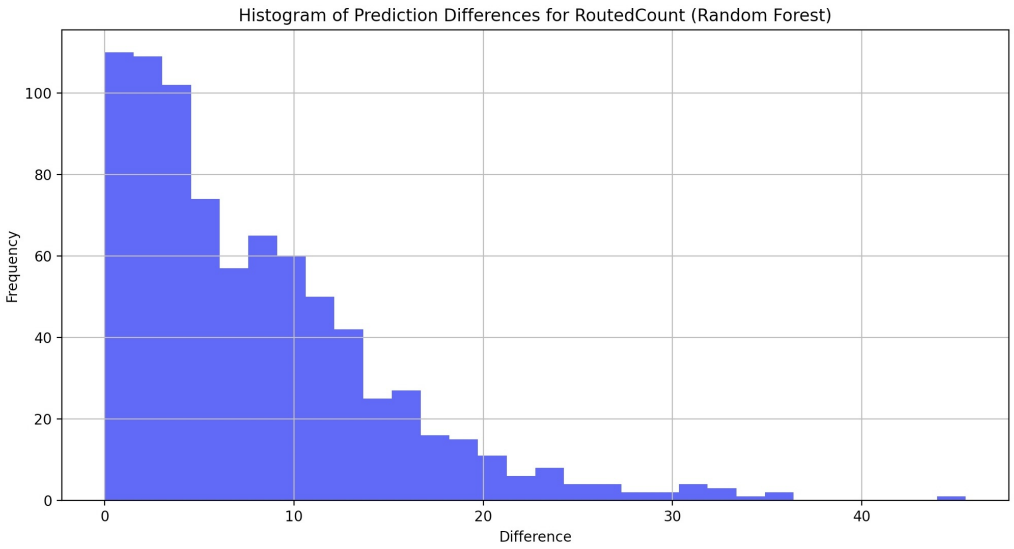


Figure 5.18

Now, the differences have significantly dropped but still occupy unwanted values. Ideally, we would like most predictions to have a difference with the actual data smaller than 10, with some being in the space between 10 and 15. This would be a particularly narrow margin which would then minimise the errors in the drivers' shifts.

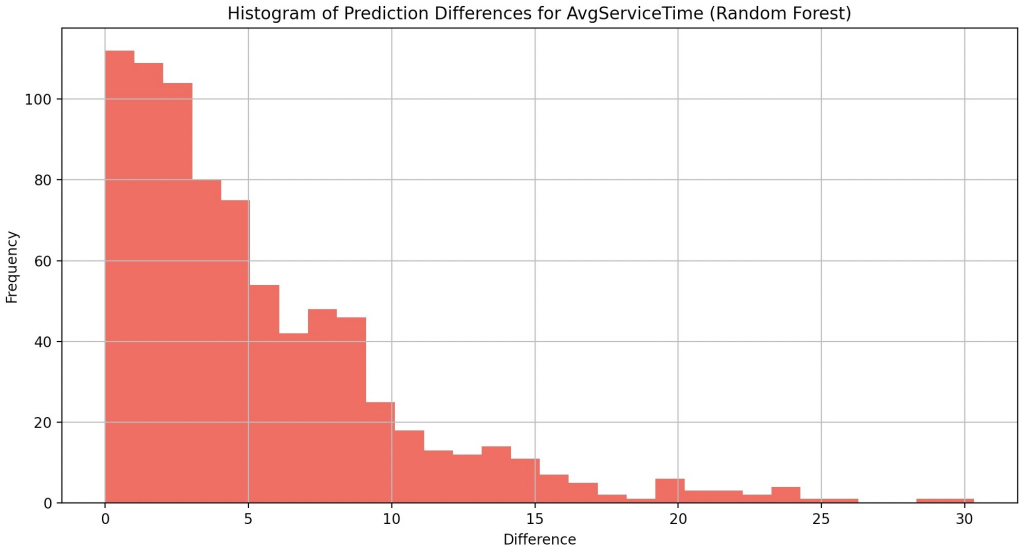


Figure 5.19

Having tested a wide variety of models and reached a threshold as to how much the performance can further be improved we need to start thinking outside the box. The basic logic behind model training in this chapter, has been that the outcome variables solely depend on the pair (M,P). But if we carefully consider the fundamentals of the problem, there are a lot of other factors contributing to the result. Being a road assistance service, traffic, time of the day, the incidents that occurred in the

previous hour and even the weather play an important role. In the next chapter, we will try to enter all these parameters in the model training process without losing significant computational speed. As the number of parameters increases, so does the danger of overfitting, so we will need to be extremely cautious as to the nature and the amount of training data utilised.

Chapter 6

Model Optimization: Parameters and Edge Cases

Every professional solution at one point needs to be fine-tuned. Drawing from the previous chapter, this is definitely the case with ours too. Here, we will restructure the solution in order for the models to have higher adaptability to the training data without losing the general context which the problem introduces in the first place. The aforementioned edge cases will be analogously handled by adding new parameters in the model training process. Finally, via the technique of elimination, we will come up with a final solution to the approximation of the general problem.

6.1 A More Holistic Approach

Let's make clear that there are two basic philosophies to approach the solution. The first one, involves training our models based on (only) two parameters the pair (M,P). This is the approach that we took on chapter 3 and has the advantages of being quicker and more precise in terms of chronological order. We can run the model more times in order to get the clearest results possible. But it is obvious that this method even when numerous models are tested, cannot provide the best feasible solution. If we want to achieve a better mixture, then we must reconsider the basic pillars of our first plan.

The second, proceeds towards a more holistic review that takes into account various other deciding factors like the time of day, the traffic, the total incidents happening in that specific hour as well as a new heaviness metric that we will discuss later. Of course, we could not keep every parameter in the final model training since that would make it prone to overfitting. For instance, the time of day can indirectly include the traffic so following simple logic and trial and error techniques we came up with the final solution. While this can be a more time/resource consuming method, it does not exceed any forbidding limits. In other words, this is a price that we can afford to pay for much improved prediction quality.

The available data on behalf of the simulator (5.1), was too vast for our models. The solution included isolating the most important information and transforming it into one big .json file that could be used as a training dataset.

```
{
    "TotalM": 5,
    "TotalP": 6,
    "RoutedCount": 12,
    "TotalV": 11,
    "AvgServiceTime": 30.704166666666666,
    "NonEmptyLines": 43,
    "hour": 6,
    "lastFreeV": 7,
    "HeavinessMetric": 0.36363636363636365
},
```


Figure 6.1: JSON object inside the dataset.

Now, we used the same models that proved the most capable in the previous chapter but with the updated dataset. Once again the training scripts were written in Python, taking advantage of the various operations that the *scikit – learn* library offers. This approach significantly improved the solution and increased the performance metrics while maintaining the generality.

6.1.1 Total Incidents

A deciding factor in the correct training of the models is the number of total incidents that occurred inside the hour of concern. This is a crucial parameter since it can lead to many identifiable edge cases if not properly handled. For a more clear picture, we will take a more practical route of explanation. Even though the simulator produces realistic combinations of (M,P), there can be significant differences between simulations. For instance, if for two different simulations there is a combination $(M, P) = (90, 90)$, it is imperative that we take into account the number of incidents that occurred during those simulations. Else, we will get two different results for the same resource pair and thus confuse the model. This can lead to significantly lower accuracy rates and can damage the final prediction of the system.

6.1.2 Time of Day

We should keep in mind that this model training has been done for predicting road assistance services in the streets of the general Athens, Greece metropolitan area. The traffic can notably vary depending on different times of the day. Just like the case of the total incidents, it would be totally unfair to make the same prediction for 06:00 AM and 02:00 PM, as the first one has generally lower traffic congestion problems than the second. So adding the specific time-frame to each of the objects, where each represents a single simulation, adds unique value and a different perspective to the way the model learns. We do not need to add any other relevant information since the data is big enough for the machine learning model to recognise the pattern of decreasing service ability during the rush hour period, compared to the rest of the day.

6.1.3 Heaviness Metric

Experimentally, despite all of the fine-tuning done above there was still room for improvement. The main concern of our first solution were the edge cases that appeared, and the only way that they could be addressed was if we delved deeper into the complexities of the problem. Indeed, one very important factor that has not yet been taken into account is the fact that service vehicles can remain occupied after the end of an hour. For example, if an incident occurs at 02:55 PM and the average service time of that hour is 35 minutes, then the respective vehicle will be occupied until about 03:30 PM. So we will be entering the next hour with lower resources than expected. Because we have not transitioned to the state of the problem where we take into account continuous pairs of hours (as it would take a truly complicated process to do so), we need to create a "heaviness" metric of each hour which translates with as much precision as possible the load of an hour's requests to a single numeric value. For this reason, we capitalized on a data point referring on the free vehicles at the end of each hour (simulation). So according to the following formula:

$$\text{adjusted\_heaviness} = \frac{\text{occupied\_vehicles\_ratio} + \text{avg\_etap}}{2} \tag{6.1}$$

where

$$\text{Occupied Vehicles Ratio} = \frac{\text{OccupiedVehicles}}{\text{TotalVehicles}} \tag{6.2}$$

Where *TotalVehicles* is the number of vehicles at the beginning of the hour and *OccupiedVehicles* is the number of the "busy" ones at the end respectively. Moreover, the *Avgetap* value is the average estimated availability time. One of the columns in the excel datasheets available, has the estimated availability time of each vehicle in the form of a one-dimensional array. By taking the average of these times and adding plugging the number into the equation 6.1, we can then take a both reliable and influential to the final prediction heaviness metric. Ultimately, this equation intuitively measures the proportion of available resources that are being utilized at any given time. In other words, it directly reflects on the operational load of the system. Especially in such a scenario where resource allocation and efficiency are crucial operational considerations, this metric can prove to be extremely useful in the detection of bottleneck cases in the system. Furthermore, we needed a computationally efficient mathematical equation that can allow real-time data analysis and integration into live systems.

6.2 Data Analysis

In our whole process, we have been using data in json format since it has proven to be the most efficient and reliable method. Two general methods have been followed. The first basically preexisted in the system's solution, but it proved necessary for a second one, more holistic to be created. These two needed two different training datasets, each one with its own peculiarities. Generally, in machine learning the models besides their own algorithmic structures are really dependant upon the respective training data that they are given. So in numerous cases, the difference between various model performances is mostly due to dataset completeness issues.

It should be noted that in this section the machine learning model that is being trained is the Random Forest Regressor. Based on the previous chapters, its increased performance and running time efficiency, it proved to be the most suitable option for our problem.

In the first case, as explained above we followed the logic that the number of incidents and average service time mostly depends on the pair (M,P) and its various values. This logic requires a json file with numerous objects, each with different simulation outcomes. If we wanted to predict the "RoutedCount" for a particular hour of the day, we fed to the model numerous "RoutedCount" values, each corresponding to a different (M,P) combination. Then, based of course on its respective algorithm, the model could find patterns in the data and predict the value for the rest of the combinations. The json objects given in this particular case had the following form:

Figure 6.2: JSON object in the beginning of the training.

Depending on the value we wanted to predict between "RoutedCount" and "AvgServiceTime", we made sure that the model takes the right one into account in order to make as accurate a prediction as possible. The next step was the evaluation of the aforementioned predictions. As explained in , the main performance metrics used were the R^2 score and the MSE. While reliable, they can be overly influenced by the quality of the data and hence could not be the only source of a metric. It was evident that we had to create our own metrics, especially if we needed to specifically locate problems like various edge cases and deal with them accordingly.

Thus, using cross-validation we drafted a python script that used the 80% of the dataset to train the model and the rest 20% to test its answers. It compared the predictions with the corresponding values generated by the simulator, therefore enabling us to locate the different edge cases and find the reasons creating them. The product of the above process was a json file with objects containing information about the difference of the aforementioned pairs of values. Then, empirically and using once again the standard performance metrics of R^2 score and MSE, we pinpointed the exact difference above which the prediction was considered an edge case. For the RoutedCount it proved to be 10, while for the AvgServiceTime 15.

```
{
    "TotalM": 44,
    "TotalP": 85,
    "Actual RoutedCount": 98,
    "Predicted RoutedCount": 92.15396031746032,
    "Difference in RoutedCount": 5.846039682539683,
    "Actual AvgServiceTime": 29.750170068027213,
    "Predicted AvgServiceTime": 32.82669222425755,
    "Difference in AvgServiceTime": 3.076522156230336
},
```


Figure 6.3: JSON object in the final stage of the training.

This same test was applied on both the first and the second method, with the latter performing significantly better. Its more holistic view allowed us both to train and run the model fewer times chronologically compared to the first and get more reliable predictions out of it. The logic is a bit more complex but still simple, compared to other options. Once again, it proved that a simple solution is enough in order to solve various practical methods. Out of the various available information generated by the simulation, through an elimination process we isolated the ones that most significantly affected the prediction and created the final dataset. Before delving into that though, we will first analyse the elimination process.

As explained briefly in 5.1 the simulation is a rather complex function which can almost ideally emulate real-world conditions. As a result, out of each run a Microsoft excel sheet is generated that contains various information about the details of each incident and its service needs. Out of this data, a lot of features useful to the training process can be picked up. For instance, whether or not the vehicles need to be transfered by the insurance company, the exact time of the incident, whether it has been serviced, the total number of available service vehicles at the moment of the incident or the approximate time that each occupies vehicle will become available.

At first, one might be tempted to give an active role to each and every aforementioned feature. This method would actually give us extremely good performance indexes when testing our model on the same dataset that it was trained but it would make it particularly prone to overfitting. This means that when tested on any other dataset its performance would significantly decrease, therefore making it practically useless. The whole logic is to find the turning point in which the features are plenty so that the model can obtain a good understanding of the peculiarities of the data structure while at the same time maintaining the generality needed in its predictions.

6.3 Feature Elimination and Overfitting Handling

The challenge described in the previous section can be addressed via a process called feature elimination. It basically involves testing all the relevant and possibly useful features in the model training and through various stages eliminating a percentage of them until a point is reached in which the model performs well both on the dataset it was trained on and other datasets. So, in order to properly evaluate each feature's importance, we needed to create a python script that would train the models properly and perform the necessary feature quantifications [46]. After training, the script retrieves and prints the feature importances from both models.

In the Random Forest Regressor, feature importances are calculated based on how much each feature decreases the weighted impurity in a tree. More specifically in the context of regression, impurity may be measured by variance, and a feature is considered more important if it reduces the variance more when it is split upon. The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. It is also known as the Gini coefficient and it is further analyzed in Appendix B.

Initially, we had numerous features taking part in the training process. Through the aforementioned script and its evaluation metrics we managed to eliminate each feature that had a Gini importance lower than 0.05. The above value was determined empirically. The output of the script had the following form for the two variables whose values we have been trying to accurately predict. The printed feature importances tell us how much each feature contributes to the model's predictions. A higher value means the feature is more important for making accurate predictions. By following these steps, the script leverages the Random Forest Regressor model to understand the relationship between the features and each target variable in your dataset and identifies which features are most influential in predicting the outcomes.

```
Feature Importances for RoutedCount:
TotalM: 0.07412581454578812
TotalP: 0.30950286185260173
NumberOfIncidents: 0.5491155731916247
HourOfDay: 0.05585244567965207
AdjustedHeaviness: 0.011403304730333404

Feature Importances for AvgServiceTime:
TotalM: 0.1383359641795571
TotalP: 0.10763230285190084
NumberOfIncidents: 0.44514421410533245
HourOfDay: 0.22845374186795422
AdjustedHeaviness: 0.08043377699525538
```

Figure 6.4: Feature Importances

As we can clearly see, because the model adapts to the patterns that each hour's data hides, it values significantly the Number of Incidents. We say each hour's patterns because the Number of Incidents mostly depends on whether it is a rush hour or other kind of time of the day. Through the above process the final dataset occurred which comprises of the following json objects:


```
{
    "FileName": "Hour20Day14ID83IDX6.xlsx",
    "RoutedCount": 50,
    "TotalM": 29,
    "TotalP": 48,
    "NumberOfIncidents": 151,
    "AvgServiceTime": 36.156666666666666,
    "AdjustedHeaviness": 0.08435420911791422,
    "HourOfDay": 20
},
```


Figure 6.5: Final form of the json dataset.

The "FileName" attribute is not a feature and is not taken into account during model training. The reason for its existence in the objects is purely for our own facilitation so that there can be proper proof-reading and object-file cross reference when needed.

Although the above results are convincing and the methods used to test the validity of the results are widely known and solid, we still need another form of safety net in order to make the final feature choice. This came in the form of different cross-validation methods that provided a more holistic view of the nature of the predictions. The most prominent one was the k-fold cross-validation which actually enabled us to fully take advantage of the existing datasets. This proved to be extremely useful in the long run since the production of different datasets through new simulations is costly. Below lie the results of the k-fold cross validation. As it is clear, for each iteration (fold) there is a different result for both the performance metrics and the Gini coefficients.

```
Fold 1
RoutedCount - MSE: 23.607488565366467, R2: 0.9604152379781368
AvgServiceTime - MSE: 16.34509903775345, R2: 0.8132865276767174

Gini Coefficients for RoutedCount:
TotalM: 0.0759
TotalP: 0.3088
NumberOfIncidents: 0.5512
HourOfDay: 0.0525
AdjustedHeaviness: 0.0116

Gini Coefficients for AvgServiceTime:
TotalM: 0.1399
TotalP: 0.1113
NumberOfIncidents: 0.4417
HourOfDay: 0.2243
AdjustedHeaviness: 0.0827
```

Figure 6.6: Iteration 1

```
Fold 2
RoutedCount - MSE: 20.679694443607985, R2: 0.9641337606006609
AvgServiceTime - MSE: 18.807387286589982, R2: 0.7868957850898134

Gini Coefficients for RoutedCount:
TotalM: 0.0768
TotalP: 0.3022
NumberOfIncidents: 0.5568
HourOfDay: 0.0523
AdjustedHeaviness: 0.0119

Gini Coefficients for AvgServiceTime:
TotalM: 0.1309
TotalP: 0.1063
NumberOfIncidents: 0.4401
HourOfDay: 0.2364
AdjustedHeaviness: 0.0862
```

Figure 6.7: Iteration 2

```
Fold 3
RoutedCount - MSE: 20.10116987481257, R2: 0.9644738434512605
AvgServiceTime - MSE: 16.851683620736512, R2: 0.8040983163262416

Gini Coefficients for RoutedCount:
TotalM: 0.0737
TotalP: 0.2928
NumberOfIncidents: 0.5677
HourOfDay: 0.0545
AdjustedHeaviness: 0.0113

Gini Coefficients for AvgServiceTime:
TotalM: 0.1463
TotalP: 0.1032
NumberOfIncidents: 0.4520
HourOfDay: 0.2260
AdjustedHeaviness: 0.0725
```

Figure 6.8: Iteration 3

```
Fold 4
RoutedCount - MSE: 21.56814110348195, R2: 0.9609756439465476
AvgServiceTime - MSE: 15.348943399357582, R2: 0.8057996924170312

Gini Coefficients for RoutedCount:
TotalM: 0.0779
TotalP: 0.2987
NumberOfIncidents: 0.5570
HourOfDay: 0.0555
AdjustedHeaviness: 0.0109

Gini Coefficients for AvgServiceTime:
TotalM: 0.1429
TotalP: 0.1019
NumberOfIncidents: 0.4584
HourOfDay: 0.2197
AdjustedHeaviness: 0.0770
```

Figure 6.9: Iteration 4

```
Fold 5
RoutedCount - MSE: 23.23470315933132, R2: 0.960806215592592
AvgServiceTime - MSE: 16.044811776977074, R2: 0.7829656307550263

Gini Coefficients for RoutedCount:
TotalM: 0.0797
TotalP: 0.2881
NumberOfIncidents: 0.5727
HourOfDay: 0.0481
AdjustedHeaviness: 0.0115

Gini Coefficients for AvgServiceTime:
TotalM: 0.1330
TotalP: 0.1019
NumberOfIncidents: 0.4571
HourOfDay: 0.2305
AdjustedHeaviness: 0.0775
```

Figure 6.10: Iteration 5

It is evident that while the performance values variate in each iteration, the change in value is not significant and therefore gives us a good understanding of how the

model has obtained a generality in terms of the data it is being tested upon. Generally, the case is that the number of total platforms is more influential than the respective number of motorcycles. This is logical, because despite of the fact that the motorcycles can navigate through traffic and can serve an incident in less time, only a certain fraction of the daily incidents can be routed to them. Duo to the nature of the incidents, whether it is a crash or a mechanical failure, a vast amount of them -possibly the majority- needs to be carried by a platform. This conclusion is supported by the overwhelmingly bigger Gini coefficient of the *TotalP* feature when compared to the respective *TotalM*'s value.

Furthermore, while the Gini Coefficient for the *AdjustedHeaviness* feature is steadily low, we have concluded to the necessity of this feature as it eliminates certain dangerously extreme edge cases. This kind of feature, the one that eliminates a very specific problem inside our solution must be dealt with patience. While every solution will have imperfections, the key is to find the balance between dealing with the ones that really pose a threat to the solution's correctness and maintaining its generality. The latter simply translates to avoiding overfitting. After all the testing and validation steps, we arrived to the point in which a definitive set of results can be presented.

6.4 Experimental Results

Following our taking of all of the aforementioned methods, algorithms and models into careful consideration and a prodigious amount of trial and error tests, this section corresponds to the formal presentation of the experimental results. We will demonstrate the final predictions of the most prominent models, the compelling reasons for the choice we made and the techniques we used to identify the edge cases in every prediction.

Firstly, let's clarify that a number of models could have been the final one. The reason behind our choice is due to a holistic approach to both the advantages and disadvantages of each one. The performance metrics displayed below were merely a formal way of showing the effectiveness of each model. Using the data format that was analyzed in this chapter, we trained the non linear models of chapter 3 and received the following results:

```
RoutedCount - MSE: 169.96169400084045 , R2: 0.7181670737484604
AvgServiceTime - MSE: 34.14642877694796 , R2: 0.6057004001766506
```

Figure 6.11: Performance metrics for the two Gaussian Process models.

It will be noticeable from the next results that the Gaussian Process' adaptability to data is significantly lower than the other models. This can be attributed to several reasons, each inherent to the nature and mathematical foundation of Gaussian Processes. Unlike Random Forest or Gradient Boosting, which are deterministic, Gaussian Process models the distribution over possible functions that fit the data. This allows it to not only make predictions but also estimate the uncertainty in these predictions, potentially providing more information and accuracy especially in

regions where data is sparse or the function behavior is complex. Additionally, due to its probabilistic foundation and the way Gaussian Process regularizes the estimation of the function that fits the data (by maximizing the marginal likelihood), Gaussian Process can be less prone to overfitting compared to models that might aggressively minimize prediction error on the training data without adequately accounting for model complexity.

When it comes to the model's prediction accuracy based on our own training data, we suggest that it is lower mainly because of the dataset's straightforward nature. There simply are not enough peculiarities in the data structure to give a measurable advantage to the Gaussian Process when compared to its simpler counterparts. So, the ease of interpretation and the straightforward nature of models like Random Forest and Gradient Boosting can be an advantage.

```
RoutedCount - MSE: 27.81101260911441 , R2: 0.9538833787711811
AvgServiceTime - MSE: 17.143815132010126 , R2: 0.8020349510001885
```

Figure 6.12: Performance metrics for the two Gradient Boosting models.

Clearly, Gradient Boosting outperforms the Gaussian Process. This can not only be interpreted by the performance metrics, but by the time efficiency and the nature of its predictions. Further explanation will be given shortly.

```
RoutedCount - MSE: 49.227000000000004 , R2: 0.9183710803652267
AvgServiceTime - MSE: 32.3698419676562 , R2: 0.6262152092810244
```

Figure 6.13: Performance metrics for the two K-Neighbors models.

The K-Neighbors algorithm is the one currently being used in the system. While fast and precise, it still left some edge cases mishandled. Also, while the speed of the model is important, small differences are not game-changers in the decision making process. Our datasets have a big size but are not that sizeable compared to others. Thus, while we need a fast model, if one is a fraction slower but offers improved accuracy we tend to opt for it.

```
RoutedCount - MSE: 23.714182950954875 , R2: 0.9606768006519226
AvgServiceTime - MSE: 16.728645928484752 , R2: 0.8068290409437773
```

Figure 6.14: Performance metrics for the two Random Forest models.

In order to make the final choice between the models we developed a new python script which indicated the edge cases in each of the predictions. When using k-fold validation, we tried to print for each sub-dataset the edge cases of the prediction and the objects those corresponded to. This method enabled us to categorize the objects that caused these loosely false predictions and find ways to eliminate them. When there was a clear connection between the objects that are created by the edge cases we tried to battle it by inserting a specific feature. This is the reason why we

created the *AdjustedHeaviness* feature and deleted some others. Finally, we reached a point with limited edge cases that did not have any specific connection between them. Empirically, in the aforementioned script we created a couple of thresholds, one for the *RoutedCount* variable and the other for the *AvgServicetime*. Their values were:

$$Threshold1 = 10 \tag{6.3}$$

and

$$Threshold2 = 15 \tag{6.4}$$

for the Routedcount and AvgServicetime respectively. Thus, we received the below outcome:


```
Edge Cases for AvgServiceTime with entire objects and differences:
{
    "FileName": "Hour12Day14ID7IDX9.xlsx",
    "RoutedCount": 65,
    "TotalM": 24,
    "TotalP": 27,
    "NumberOfIncidents": 336,
    "AvgServiceTime": 37.41564102564102,
    "AdjustedHeaviness": 0.22453796443066418,
    "HourOfDay": 12
}
Difference in AvgServiceTime: 31.913847539153224
```


Figure 6.15: Edge case for the AvgServiceTime variable.


```
Edge Cases for RoutedCount with entire objects and differences:
{
    "FileName": "Hour12Day6ID109IDX5.xlsx",
    "RoutedCount": 62,
    "TotalM": 9,
    "TotalP": 33,
    "NumberOfIncidents": 283,
    "AvgServiceTime": 33.74865591397849,
    "AdjustedHeaviness": 0.3163737362912157,
    "HourOfDay": 12
}
Difference in RoutedCount: 12.807910052910081
```


Figure 6.16: Edge case for the RoutedCount variable.

So compared to the histograms given in chapter 3, the following result has occurred after all the optimizaton methods discussed above. It shows a significant improvement.

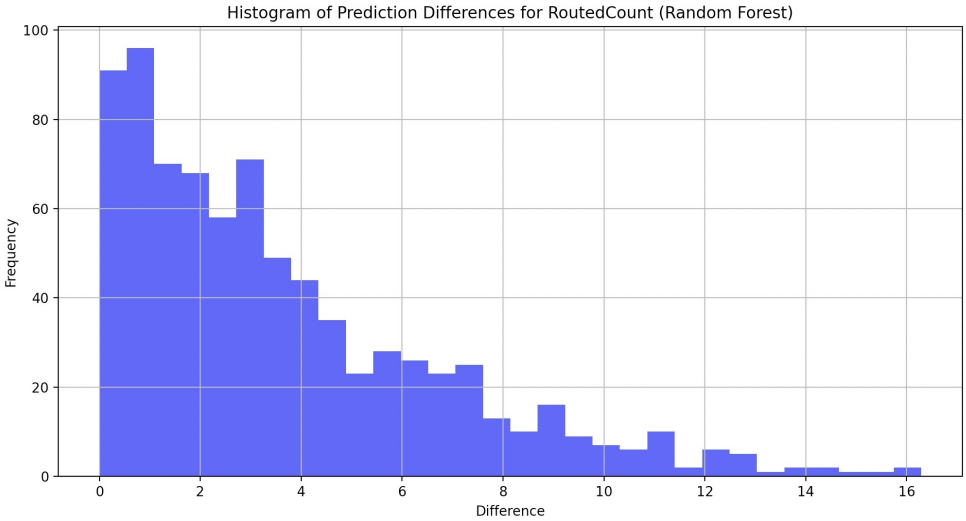


Figure 6.17

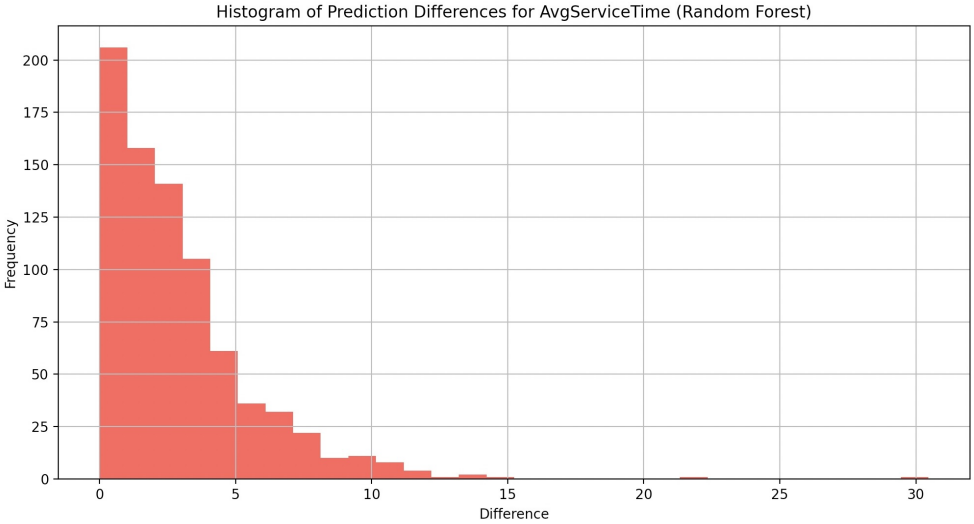


Figure 6.18

All in all, with this holistic approach in the training process we have radically ameliorated the quality of the predictions made by the models. The last two graphs have come from the training of the Random Forest Regressor, which proved to be both the most accurate and the most efficient model. The second closest was the gradient boosting, which while it can offer excellent solutions, has on average a bit more edge cases than the random forest.

At this point, we would like to point out the special case of the Neural Networks and the Multi-Layer Perceptron (MLP). While it is particularly accurate, it is prone to overfitting and due to its complexity, when tested on different datasets its performance dropped significantly. This is because the MLPs can have multiple layers with a large number of neurons, giving them a high capacity to learn complex patterns [47]. They can end up memorizing the training data, including noise and outliers, rather than learning the underlying patterns that generalize well to unseen data.

Moreover, they are sensitive to the scale of the input features, and without proper feature scaling, they might focus disproportionately on features with higher numeric ranges, potentially leading to overfitting on those features if they do not represent meaningful aspects of the problem. So, despite their vast capabilities, one would say that these kind of models are "overqualified" for our problem.

Chapter 7

The Case for OPL: Streamlining Complex Vehicle Routing and Capacity Management Tasks

Throughout this thesis, the focus has been on optimizing the quality of the machine learning model's prediction which in turn provides the foundation for the scheduling solver. As described in chapter 5, the last part of the algorithm consists of an integer program which addresses the vehicle routing problem. It is essentially a complex constraint programming problem and the solution is written in python, effectively utilizing the functionalities of the OR-Tools. In this chapter we will focus more on this last section and how it can be optimized. It will be proven that Optimization Programming Language (OPL) is a superior tool for solving complex vehicle routing and capacity management problems, compared to the widely used OR-Tools framework. Through a detailed case study, this chapter illustrates how OPL not only enhances model clarity and maintainability but also optimizes performance in solving large-scale optimization problems. We delve into a direct translation of a previously implemented OR-Tools model into OPL, highlighting the differences in code complexity, readability, and execution efficiency.

7.1 The Role of OR-Tools in the Current Vehicle Routing Solution

In operations research and optimization, modeling plays a crucial role in formulating real-world problems into mathematical representations that can be solved computationally. OR-Tools, an open-source software suite developed by Google, provides a comprehensive platform for modeling and solving combinatorial optimization problems. It supports a wide range of problem types, including linear programming (LP), mixed-integer programming (MIP), constraint programming (CP), and vehicle routing problems (VRP). By leveraging powerful solvers and a flexible API, OR-Tools enables researchers and practitioners to define constraints, objectives, and decision variables with ease, facilitating the development of efficient and effective optimization models.

A typical modeling process in OR-Tools begins with the definition of decision variables, which represent the choices to be made. These variables are subject to constraints, reflecting the problem's limitations or requirements, and an objective function that quantifies the goal, such as minimizing cost or maximizing profit. OR-Tools provides an intuitive interface for specifying these components using various programming languages, including Python, C++, and Java. Once the model is defined, OR-Tools employs state-of-the-art algorithms and solvers to find optimal or near-optimal solutions, offering robust performance and scalability. The modular architecture of OR-Tools also allows for the integration of custom heuristics and optimization techniques, making it a versatile tool for tackling complex optimization challenges in various domains.

This script begins by generating a dataset of incident data using the generate incident data function. This function simulates the number of incidents occurring between the hours of 6 AM and 10 PM. The incident frequency varies by hour, reflecting typical patterns such as lower incidents early in the morning, a peak during working hours, and a decline in the evening. For each hour, the script calculates the number of incidents, the proportion routed (assumed to be 70%), and generates random values for the number of available motorcycles and platforms. The generated data is stored in a list of dictionaries, with each dictionary representing an hour's data.

The core of the script is the *capacitymodel* function, which employs Google's OR-Tools Constraint Programming (CP) solver to optimize resource allocation. This function defines decision variables for the start and total counts of motorcycles and platforms, as well as the number of incidents that can be served each hour. Constraints are added to ensure that the served incidents do not exceed the available resources and that a minimum service level of 60% of incidents is maintained. Additionally, the script includes constraints to manage the shifts of motorcycles and platforms, ensuring that resources are correctly assigned and utilized across the hours.

The optimization objective is to minimize the total number of motorcycles and platforms used throughout the day. The CP model sums the resources used each hour and minimizes this total. Upon solving the model, the script prints the solver's runtime and checks if a feasible solution is found. If a solution is identified, it is formatted into a dictionary and then converted to a pandas DataFrame. Finally, this DataFrame is exported to an Excel file named *'capacitysolution.xlsx'*, providing a detailed record of the optimized resource allocation and incident management across the hours. This structured approach ensures an efficient and effective distribution of resources, crucial for operational planning in scenarios with fluctuating demands.

In this section, we will replace the aforementioned script, as depicted in the provided screenshots 7.4, with a model developed using IBM's Optimization Programming Language (OPL). OPL is known for its expressive power and specialized syntax designed specifically for modeling optimization problems. By translating our existing constraint programming (CP) model into OPL, we aim to leverage OPL's robust capabilities and potentially achieve more concise and readable formulations. The transition will involve defining the same decision variables, constraints, and objective functions within the OPL framework, ensuring that the logical structure and

intent of the original Python-based model are preserved.

```python
import json
import pandas as pd
from ortools.sat.python import cp_model
import random
import time

def generate_incident_data():
    data = []

    for hour in range(6, 23):  # From hour 6 to hour 22 (inclusive)
        if hour == 6:
            number_of_incidents = random.randint(0, 30)  # Minimal incidents at 6am
        elif hour in [7, 8]:
            number_of_incidents = random.randint(30, 100)  # Under 100 incidents for 7-8am
        elif 9 <= hour <= 18:
            number_of_incidents = random.randint(150, 300)  # Higher incidents from 9am to 6pm
        elif 19 <= hour <= 20:
            number_of_incidents = random.randint(100, 150)  # Moderate incidents from 7pm to 8pm
        else:  # 21 and 22
            number_of_incidents = random.randint(0, 100)  # Minimal incidents from 9pm to 10pm

        routed_count = int(0.7 * number_of_incidents)  # Assume 70% are routed
        total_m = random.randint(10, 50)  # Random number of motorcycles
        total_p = random.randint(10, 50)  # Random number of platforms

        entry = {
            "HourOfDay": hour,
            "RoutedCount": routed_count,
            "TotalM": total_m,
            "TotalP": total_p,
            "NumberOfIncidents": number_of_incidents
        }
        data.append(entry)
    return data

def capacity_model(data):
    n_hours = len(data)
    ids = list(range(6, 23))  # Hours from 6 to 22

```

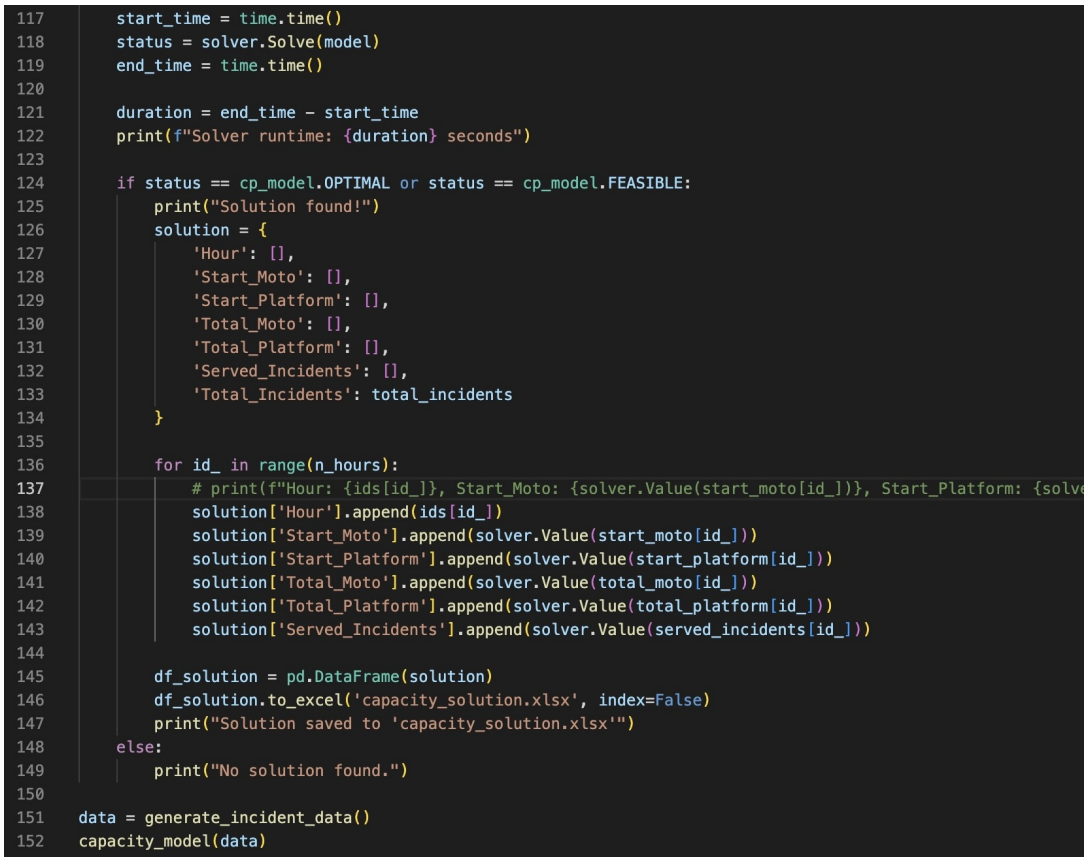
Figure 7.1: Constraint programming in OR-Tools.

```python
    model = cp_model.CpModel()

    # Decision variables
    start_moto = [model.NewIntVar(0, 100, f'start_moto[{id_}]') for id_ in ids]
    start_platform = [model.NewIntVar(0, 100, f'start_platform[{id_}]') for id_ in ids]
    total_moto = [model.NewIntVar(0, 100, f'total_moto[{id_}]') for id_ in ids]
    total_platform = [model.NewIntVar(0, 100, f'total_platform[{id_}]') for id_ in ids]
    served_incidents = [model.NewIntVar(0, 1000, f'served_incidents[{id_}]') for id_ in ids]

    total_incidents = []

    # Add constraints for each hour
    for entry in data:
        hour = entry['HourOfDay']
        number_of_incidents = entry['NumberOfIncidents']
        total_incidents.append(number_of_incidents)

        hour_index = ids.index(hour)

        # Capacity constraint: served incidents must be within available capacity
        model.Add(served_incidents[hour_index] <= total_moto[hour_index] + total_platform[hour_index])

        # Incident bound: served incidents must not exceed actual incidents
        model.Add(served_incidents[hour_index] <= number_of_incidents)

        # Minimum service level: at least 60% of incidents must be served
        model.Add(served_incidents[hour_index] >= int(0.6 * number_of_incidents))

    # Variables to track active shifts
    active_moto = [model.NewIntVar(0, 100, f'active_moto[{id_}]') for id_ in ids]
    active_platform = [model.NewIntVar(0, 100, f'active_platform[{id_}]') for id_ in ids]

    # Constraints for each hour
    for hour in ids:
        hour_index = ids.index(hour)

        # Total active motorcycles and platforms should reflect shift starts and ends
        if hour_index == 0:
            model.Add(active_moto[hour_index] == start_moto[hour_index])
```

Figure 7.2: Constraint programming in OR-Tools.

```
            model.Add(active_platform[hour_index] == start_platform[hour_index])
        else:
            previous_hour_index = hour_index - 1
            start_shift = model.NewIntVar(0, 100, f'start_shift_moto[{hour}]')
            end_shift = model.NewIntVar(0, 100, f'end_shift_moto[{hour}]')
            start_shift_platform = model.NewIntVar(0, 100, f'start_shift_platform[{hour}]')
            end_shift_platform = model.NewIntVar(0, 100, f'end_shift_platform[{hour}]')

            # Start and end of shift constraints for motorcycles
            if hour + 8 < 23:
                model.Add(start_shift == start_moto[hour_index])
                model.Add(end_shift == start_moto[hour_index - 8])
                model.Add(active_moto[hour_index] == active_moto[previous_hour_index] + start_shift - end_shift)
            else:
                model.Add(active_moto[hour_index] == active_moto[previous_hour_index] + start_moto[hour_index])

            # Start and end of shift constraints for platforms
            if hour + 8 < 23:
                model.Add(start_shift_platform == start_platform[hour_index])
                model.Add(end_shift_platform == start_platform[hour_index - 8])
                model.Add(active_platform[hour_index] == active_platform[previous_hour_index] + start_shift_platform - end_shift_platform)
            else:
                model.Add(active_platform[hour_index] == active_platform[previous_hour_index] + start_platform[hour_index])

        # Ensure total resources are correctly assigned
        model.Add(total_moto[hour_index] == active_moto[hour_index])
        model.Add(total_platform[hour_index] == active_platform[hour_index])

    # Objective: Minimize the total number of motorcycles and platforms
    total_motos = model.NewIntVar(0, 100 * n_hours, 'total_motos')
    total_platforms = model.NewIntVar(0, 100 * n_hours, 'total_platforms')

    model.Add(total_motos == sum(total_moto))
    model.Add(total_platforms == sum(total_platform))

    model.Minimize(total_motos + total_platforms)

    solver = cp_model.CpSolver()
```

Figure 7.3: Constraint programming in OR-Tools.



```
    start_time = time.time()
    status = solver.Solve(model)
    end_time = time.time()

    duration = end_time - start_time
    print(f"Solver runtime: {duration} seconds")

    if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
        print("Solution found!")
        solution = {
            'Hour': [],
            'Start_Moto': [],
            'Start_Platform': [],
            'Total_Moto': [],
            'Total_Platform': [],
            'Served_Incidents': [],
            'Total_Incidents': total_incidents
        }

        for id_ in range(n_hours):
            # print(f"Hour: {ids[id_]}, Start_Moto: {solver.Value(start_moto[id_])}, Start_Platform: {solve
            solution['Hour'].append(ids[id_])
            solution['Start_Moto'].append(solver.Value(start_moto[id_]))
            solution['Start_Platform'].append(solver.Value(start_platform[id_]))
            solution['Total_Moto'].append(solver.Value(total_moto[id_]))
            solution['Total_Platform'].append(solver.Value(total_platform[id_]))
            solution['Served_Incidents'].append(solver.Value(served_incidents[id_]))

        df_solution = pd.DataFrame(solution)
        df_solution.to_excel('capacity_solution.xlsx', index=False)
        print("Solution saved to 'capacity_solution.xlsx'")
    else:
        print("No solution found.")

data = generate_incident_data()
capacity_model(data)
```

Figure 7.4: Constraint programming in OR-Tools.

The script leverages several key programming elements of OR-Tools, a powerful optimization library from Google, to construct and solve a constraint programming

model. The core components used include the *cpmodel* module, which provides the tools necessary to define and solve constraint satisfaction problems. Within this module, the script creates various decision variables such as *startmoto*, *startplatform*, *totalmoto*, *totalplatform*, and *servedincidents* to represent different aspects of the resource allocation problem. These variables are defined using model.NewIntVar, which specifies their range of possible values, ensuring that the solution adheres to practical constraints.

The script also employs several constraints to model the real-world requirements of the problem. For instance, the model.Add method is used to enforce capacity constraints, ensuring that the number of incidents served does not exceed the available resources and that at least 60% of the incidents are served. Shift management is handled by introducing constraints that link the number of active motorcycles and platforms between consecutive hours, taking into account shift start and end times. Additionally, the objective function, defined using model.Minimize, aims to reduce the total number of motorcycles and platforms used across all hours. The CpSolver class is then used to solve the model, and the solution is extracted and printed, demonstrating the practical application of OR-Tools in optimizing resource allocation scenarios. This approach highlights OR-Tools' flexibility and effectiveness in handling complex optimization tasks through a combination of decision variables, constraints, and an objective function.

Following the implementation of the model in OPL, we will conduct a thorough comparison of its performance against the Python-based OR-Tools implementation. This comparison will focus on key metrics such as runtime efficiency, memory usage, and scalability when solving various problem instances. By benchmarking the two approaches, we aim to evaluate whether the OPL-based model can provide faster or more efficient solutions compared to the OR-Tools model. This analysis will help determine the suitability of OPL for our specific optimization tasks and assess if it offers a significant performance advantage that justifies the switch from Python's OR-Tools.

7.2 Translation to Optimization Programming Language (OPL)

Setting up the OPL Integrated Development Environment (IDE) and translating a model from Python to OPL requires a structured approach, beginning with the installation and configuration of the necessary software. First, one must obtain IBM ILOG CPLEX Optimization Studio, which includes the OPL IDE. This software can be downloaded from IBM's official website, and an appropriate license must be acquired to unlock its full capabilities. Once the software is installed, the next step is to configure the IDE. This involves setting up the environment to ensure it recognizes the project's files and dependencies, which is crucial for efficient development and debugging.

The initial setup within the OPL IDE involves creating a new project. This process begins by selecting 'New Project' and specifying the project type, typically an 'OPL Project.' Within the newly created project, one should organize the model files and data files appropriately. For example, the main model file, often named model.mod,

and the data file, data.dat, should be placed in the project directory. Ensuring that the project is correctly set up within the IDE allows for seamless execution and testing of the model. Additionally, setting the project as the active project and correctly configuring the run configurations is essential for a successful setup.

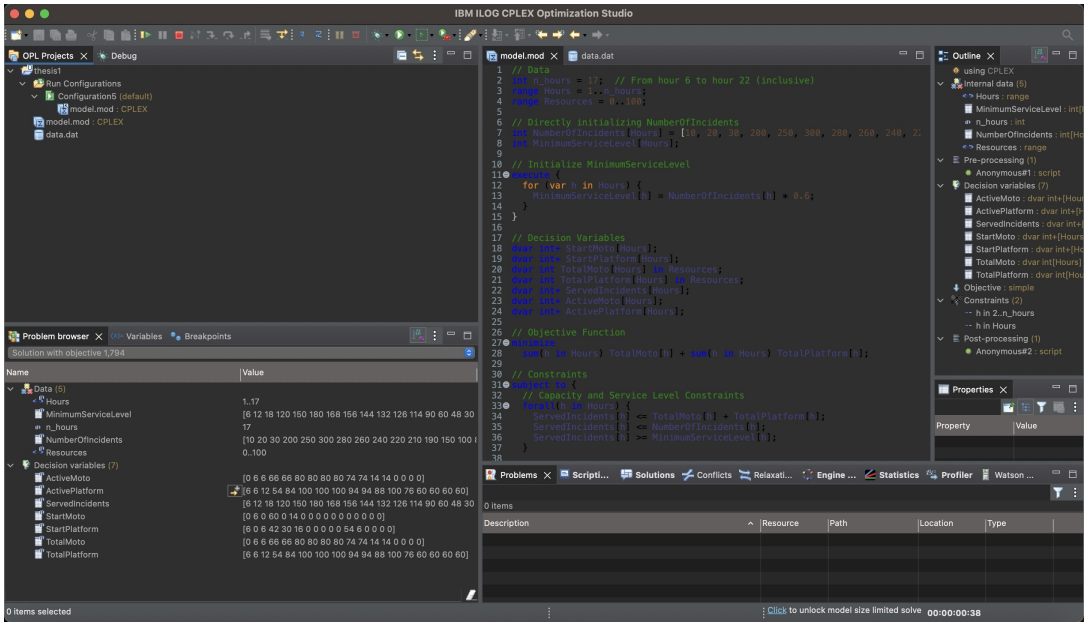


Figure 7.5: IBM ILOG CPLEX Optimization Studio

Translating the model from Python to OPL requires a careful consideration of the syntactical and functional differences between the two languages. The first step in this translation process is to understand the structure and constraints of the Python model. Key elements such as data initialization, decision variables, constraints, and the objective function must be identified and mapped to their OPL equivalents. In OPL, data is often initialized within the model file or a separate data file, and this data is then used to define decision variables and constraints. The objective function in OPL is specified using the minimize or maximize keywords, similar to the formulation in Python but adhering to OPL syntax.

During the translation, it is also important to incorporate execution blocks for post-solution analysis, similar to the use of print statements in Python. The execution block in OPL allows for actions to be performed after the solver finds a solution, such as printing results or performing additional calculations. After translating the Python model into OPL syntax, the model must be thoroughly tested within the OPL IDE to ensure it behaves as expected. This involves running the model with various data sets, checking for errors, and verifying that the constraints and objective function are correctly implemented. By systematically following these steps, one can successfully translate a Python model into OPL, leveraging the powerful features of IBM's optimization tools for enhanced problem-solving capabilities. It is additionally worth to be noted that the translation is not direct and minor changes have been made from one script to the other, as would between almost every couple of programming languages. In the following figure, we showcase the difference in program logic between python and OPL regarding the declaration of the objective.

```python
    # Objective: Minimize the total number of motorcycles and platforms
    total_motos = model.NewIntVar(0, 100 * n_hours, 'total_motos')
    total_platforms = model.NewIntVar(0, 100 * n_hours, 'total_platforms')

    model.Add(total_motos == sum(total_moto))
    model.Add(total_platforms == sum(total_platform))

    model.Minimize(total_motos + total_platforms)
```

Figure 7.6: Declaration of the objective in python.

```
// Objective Function
minimize
   sum(h in Hours) TotalMoto[h] + sum(h in Hours) TotalPlatform[h];

```

Figure 7.7: Declaration of the objective in OPL.

In both the Python and the OPL code, the objective is to minimize the total number of motorcycles and platforms used over all hours. This is achieved by first declaring two new integer variables, $total_motos$ and $total_platforms$, each with an upper bound of 100 times the number of hours, representing the maximum possible number of motorcycles and platforms. The constraints are then added to ensure that *totalmotos* equals the sum of *totalmoto* over all hours and *totalplatforms* equals the sum of *totalplatform* over all hours. Finally, the objective function is defined to minimize the sum of *totalmotos* and *totalplatforms*. On the other hand, OPL uses the minimize keyword followed by the summation of TotalMoto[h] and TotalPlatform[h] over the range of hours, effectively reducing the total resource usage across all hours. Both segments declare the variables and constraints necessary to ensure the objective function accurately represents the goal of minimizing resource usage.

7.3 Experimental Results

The generated instances for the Vehicle Routing Problem (VRP) were designed to simulate realistic traffic patterns and incident occurrences over the course of a typical day. The number of incidents was varied according to the time of day to reflect common traffic conditions. For example, at 6 am, the number of incidents was minimal, ranging between 0 and 30, as early morning traffic is typically light. Between 7 am and 8 am, incident numbers increased to between 30 and 100, capturing the beginning of the morning rush hour. From 9 am to 6 pm, the number of incidents was significantly higher, ranging from 150 to 300, reflecting peak traffic conditions throughout the day. After 6 pm, the number of incidents began to decrease, with moderate incident numbers between 100 and 150 from 7 pm to 8 pm, and dropping below 100 incidents from 9 pm to 10 pm, corresponding to the reduction in traffic as the day winds down.

The Python model, using Google's OR-Tools, was compared to IBM's CP Optimizer implemented in OPL. The performance of both solvers was evaluated based on their ability to minimize the total number of motorcycles and platforms while serving at

least 60% of the incidents in each hour. Over the 100 iterations, each solver's effectiveness was measured by the quality of the solution. The Python model demonstrated strong initial performance, quickly generating solutions and leveraging local search heuristics. However, it is advised by research papers that it often falls short in more complex instances where the number of variables is higher and the constraints more rigorous.

Below, two similar problem instances solved by both the OR-Tools and OPL solvers are visible.

Hour	Start_Moto	art_Platfor	Total_Moto	tal_Platfor	ved_Incide	tal_Incidents
6	70	0	70	0	7	14
7	0	0	42	0	25	50
8	0	0	42	0	27	55
9	0	36	42	36	78	156
10	0	41	42	77	119	238
11	0	0	42	77	113	226
12	0	0	42	77	98	196
13	28	3	70	80	150	300
14	0	0	0	80	75	151
15	0	0	0	80	80	160
16	28	0	28	80	100	201
17	0	0	28	80	108	217
18	0	0	28	80	81	162
19	0	0	28	80	54	109
20	0	0	28	80	67	134
21	0	0	28	80	39	79
22	0	0	28	80	45	90

Figure 7.8: Solution found by the python OR-Tools solver.

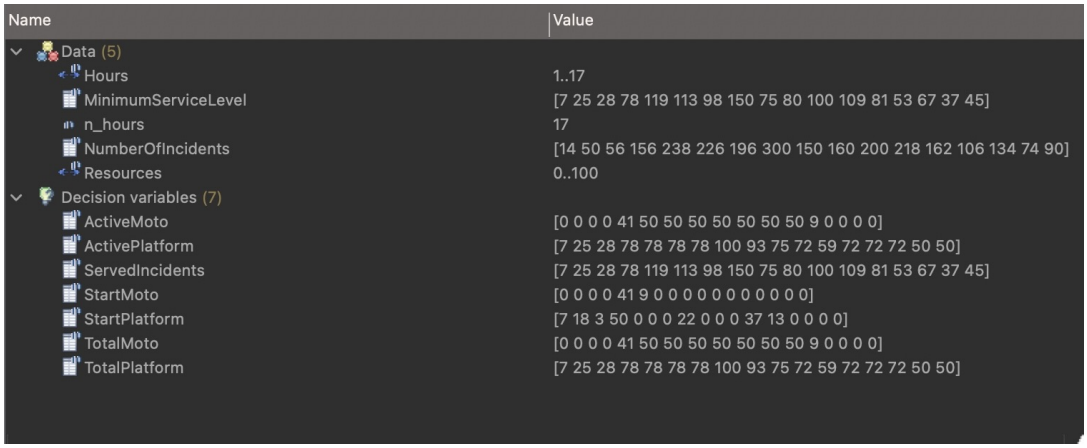


Figure 7.9: Solution found by the IBM CP Optimizer using OPL.

In comparison, the CP Optimizer consistently provided higher quality solutions in about 72% of the instances, aligning with findings from similar studies [9]. The CP Optimizer's robust optimization capabilities and ability to handle complex constraints made it particularly effective in managing the high incident volumes and

intricate resource allocation problems typical of peak traffic hours. This comparison underscores the strengths of CP Optimizer in delivering reliable, high-quality solutions for complex VRP scenarios, while also highlighting OR-Tools' proficiency in rapidly generating good solutions for simpler instances. These results validate the effectiveness of CP Optimizer for demanding optimization tasks, as demonstrated in the referenced paper.

Chapter 8

Conclusion - Future Work

In this thesis, we presented a comprehensive solution to the resource allocation problem faced by an insurance company by integrating advanced machine learning techniques with integer programming. We began with a thorough examination of various machine learning models to predict key performance indices, focusing on both linear and non-linear regression models. The decision trees and random forest models emerged as the most effective due to their ability to handle complex, non-linear relationships in the data. These models provided accurate predictions for the number of incidents each combination of motorcycles and platforms could handle, allowing for optimized deployment decisions.

To further enhance our solution, we incorporated a detailed analysis of edge cases and introduced additional parameters such as the time of day and a custom "heaviness" metric. This refinement ensured that our models could adapt to different operational scenarios and provided robust predictions across various conditions. The integration of these parameters improved the accuracy and reliability of our predictions, making the solution more adaptable and effective in real-world applications.

Our final solution involved embedding the machine learning predictions into an integer programming framework to optimize the scheduling of driver shifts. This framework accounted for business constraints, including shift lengths and holidays, ensuring compliance with industry standards and practical feasibility. The use of integer programming allowed us to derive optimal shift schedules that minimized costs while maximizing service efficiency, significantly improving the company's operational performance.

For future work, we propose further refinement of the machine learning models by exploring more advanced techniques such as deep learning and reinforcement learning, which could offer even greater predictive accuracy and adaptability. Additionally, we recommend developing a dynamic scheduling system that can adjust in real-time based on incoming data, further enhancing the responsiveness and efficiency of the resource allocation process. Continued collaboration with industry partners will also be essential to validate and refine the models in diverse operational environments, ensuring the robustness of our solution.

Bibliography

[1] El Mazgualdi, Choumicha, et al. Machine learning for kpis prediction: a case study of the overall equipment effectiveness within the automotive industry. *Soft Computing 25.4*, pages 2891–2909, 2021.

[2] Taleongpong, Panukorn, et al. Machine learning techniques to predict reactionary delays and other associated key performance indicators on british railway network. *Journal of Intelligent Transportation Systems 26.3*, pages 311–329, 2022.

[3] Shodamola, Joel, et al. *A machine learning based framework for KPI maximization in emerging networks using mobility parameters*. 2020.

[4] Manenti, Flavio. From reacting to predicting technologies: A novel performance monitoring technique based on detailed dynamic models. *Chemical Product and Process Modeling 4.2*, 2009.

[5] Solomon, Andrei, and Marin Litoiu. Business process performance prediction on a tracked simulation model. *proceedings of the 3rd international workshop on Principles of engineering service-oriented systems.*, 2011.

[6] Nakajima, Seiichi. Tpm tenkai. *Japan Institute of Plant Maintenance, Tokyo*, 1982.

[7] Ben-Hur, Asa, and Jason Weston. A user's guide to support vector machines. *Data mining techniques for the life sciences*, pages 223–239, 2010.

[8] Awad, Mariette, et al. *Support vector machines for classification*. 2015.

[9] Giacomo Da Col and Erich Teppan. Google vs ibm: A constraint solving challenge on the job-shop scheduling problem. *arXiv preprint arXiv:1909.08247*, 2019.

[10] et al. Cuvelier, Thibaut. Or-tools' vehicle routing solver: a generic constraint-programming solver with heuristic search for routing problems. *24e congrès annuel de la société française de recherche opérationnelle et d'aide à la décision.*, 2023.

[11] H. D. Block, Bruce Knight, F. Rosenblatt. Analysis of a four-layer series-coupled perceptron. ii. *Reviews of Modern Physics*, pages 1–9, 1962.

[12] Apte, Chid. The role of machine learning in business optimization. *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010.

[13] Zelinska, Snizhana. Machine learning: technologies and potential application at mining companies. *E3s web of conferences. Vol. 166. EDP Sciences*, 2020.

[14] Álvarez-Jareño, José A., Elena Badal-Valero, and José Manuel Pavía. Using machine learning for financial fraud detection in the accounts of companies investigated for money laundering. 2017.

[15] Maulud, Dastan, and Adnan M. Abdulazeez. A review on linear regression comprehensive in machine learning. *Journal of Applied Science and Technology Trends 1.2*, 2020.

[16] James, Gareth, et al. An introduction to statistical learning. vol. 112. *New York: springer*, 2013.

[17] Keith A. Marill, MD. Advanced statistics: Linear regression, part ii: Multiple linear regression. *Academic emergency medicine*, pages 94–102, 2004.

[18] Seber, George AF, and Alan J. Lee. Linear regression analysis. *ohn Wiley Sons*, 2012.

[19] Guo, Gongde, et al. Knn model-based approach in classification. *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003*, Proceedings. Springer Berlin Heidelberg, 2003.

[20] M Xu, P Watanachaturaporn, PK Varshney, MK Arora. Decision tree regression for soft classification of remote sensing data. *Remote Sensing of Environment*, pages 1–15, 2005.

[21] Segal, Mark R. Machine learning benchmarks and random forest regression. 2004.

[22] C Bentéjac, A Csörgő, G Martínez-Muñoz. A comparative analysis of xgboost. *Artificial Intelligence Review*, pages 1–20, 2021.

[23] B Jones, RT Johnson. Design and analysis for the gaussian process model. *Quality and Reliability Engineering International*, pages 1–32, 2009.

[24] Ahmadi, Saba, et al. The strategic perceptron. *Proceedings of the 22nd ACM Conference on Economics and Computation.*, 2021.

[25] Popescu, Marius-Constantin, et al. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems 8.7*, 2009.

[26] Dantzig, George B. Linear programming. *Operations research 50.1*, pages 42–47., 2002.

[27] Britannica, The Editors of Encyclopaedia. linear programming. *Encyclopedia Britannica,*, 2 Feb. 2024.

[28] Wolsey, Laurence A. *Integer programming*. 2020.

[29] Rossi, Francesca, Peter Van Beek, and Toby Walsh, eds. Handbook of constraint programming. *Elsevier,*, 2006.

[30] et al. Van Hentenryck, Pascal. Constraint programming in opl. *International Conference on Principles and Practice of Declarative Programming. Berlin, Heidelberg: Springer Berlin Heidelberg,*, 1999.

[31] et al. Stuckey, Peter J. The evolving world of minizinc. *Constraint Modelling and Reformulation*, pages : 156–170, 2007.

[32] Krzysztof R. Apt and Mark Wallace. Constraint logic programming using eclipse. *Cambridge University Press,*, 2006.

[33] Guido Tack Schulte, Christian and Mikael Z. Lagerkvist. Modeling and programming with gecode. *Schulte, Christian and Tack, Guido and Lagerkvist, Mikael 1*, 2010.

[34] et al. Carlsson, Mats. Sicstus prolog user's manual. vol. 3. no. 1. *Kista, Sweden: Swedish Institute of Computer Science,*, 1988.

[35] Thibaut Feydy and Peter J. Stuckey. Lazy clause generation reengineered. *International Conference on Principles and Practice of Constraint Programming. Berlin, Heidelberg: Springer Berlin Heidelberg,*, 2009.

[36] et al. Brand, Sebastian. Encodings of the sequence constraint. *International conference on principles and practice of constraint programming. Berlin, Heidelberg: Springer Berlin Heidelberg,*, 2007.

[37] Alan M. Frisch and Peter J. Stuckey. The proper treatment of undefinedness in constraint languages. *International Conference on Principles and Practice of Constraint Programming. Berlin, Heidelberg: Springer Berlin Heidelberg,*, 2009.

[38] Laurent Michel and Pascal Van Hentenryck. A modeling layer for constraint-programming libraries. *INFORMS Journal on Computing 17.4*, pages 389–401, 2005.

[39] Joaquín. Rodriguez. A constraint programming model for real-time train scheduling at junctions. *Transportation Research Part B: Methodological 41.2*, pages 231–245, 2007.

[40] Alexander Fay and Eckehard Schnieder. Knowledge-based decision support system for real-time train traffic control. *Computer-Aided Transit Scheduling: Proceedings, Cambridge, MA, USA, August 1997. Berlin, Heidelberg: Springer Berlin Heidelberg,*, pages 347–370, 1999.

[41] et al. Fernández, Antonio. Multi-train simulator for regulation purposes. *WIT Transactions on The Built Environment 50*, 2000.

[42] Box, G. E. All models are wrong, but some are useful. *obustness in Statistics 202.1979*, page 549, 1979.

[43] Tibshirani, Robert. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology 58.1*, pages 267–288, 1996.

[44] Reid, Stephen, Robert Tibshirani, and Jerome Friedman. study of error variance estimation in lasso regression. *Statistica Sinica*, 2016.

[45] Delashmit, Walter H., and Michael T. Manry. Recent developments in multilayer perceptron neural networks. *Proceedings of the seventh annual memphis area engineering and science conference, MAESC. Vol. 7.*, 2005.

[46] Sanasam, Ranbir, Hema Murthy, and Timothy Gonsalves. Feature selection for text classification based on gini coefficient of inequality. *Feature Selection in Data Mining. PMLR*, 2010.

[47] Rynkiewicz, Joseph. General bound of overfitting for mlp regression models. *Neurocomputing 90*, pages 106–110, 2012.

[48] Marshall H. Stone. The generalized weierstrass approximation theorem. *Mathematics Magazine 21.5*, pages 237–254, 1948.

[49] L. H. Loomis. Introduction to abstract harmonic analysis. *Courier Corporation*, 2013.

[50] Mark Krein and David Milman. On extreme points of regular convex sets. *Studia Mathematica 9*, pages 133–138, 1940.

[51] Leo. Breiman. Classification and regression trees. *Routledge*, 2017.

[52] Michael Steinbach Tan, Pang-Ning and Vipin Kumar. Introduction to data mining. *Pearson Education India*, 2016.

Appendix A

Stone Weierstrass theorem

The Stone-Weierstrass theorem [48] is a fundamental result in functional analysis that generalizes the Weierstrass approximation theorem. It states conditions under which a subalgebra of continuous functions on a compact Hausdorff space is dense in the algebra of all continuous functions.

Let S be a locally compact Hausdorff space, and let $C(S)$ denote the space of continuous complex-valued functions on S that vanish at infinity (in the non-compact case). Let E be a vector subspace of $C(S)$ that is closed under complex conjugation. The Stone-Weierstrass theorem provides conditions under which E is uniformly dense in $C(S)$.

The conditions are as follows:

1. **Separation of Points**: For each pair of distinct points P and Q in S, there exists an f in E such that $f(P) \neq f(Q)$.
2. **Vanishing at Points**: For each point P in S, there exists an f in E such that $f(P) \neq 0$.
3. **Closure Under Multiplication**: Whenever f and g are in E, the product fg is in E.

If these conditions are satisfied, then E is uniformly dense in $C(S)$.

Proof Sketch

The proof involves several key ideas from measure theory and convex analysis [49], particularly the Krein-Milman theorem [50] and properties of extreme points.

- **Reduction to Real-Valued Functions**: Since E is closed under complex conjugation, it suffices to consider real-valued functions.
- **Measure Theory Application**: Define $U(E)$ as the set of all real-valued Borel measures on S with total variation at most 1, such that the integral of any function in E with respect to these measures is zero.
- **Compactness and Convexity**: $U(E)$ is a compact, convex set. If E is not dense in $C(S)$, then $U(E)$ contains a nonzero element by the Krein-Milman

theorem, which states that any compact convex set is the closed convex hull of its extreme points.

- **Extreme Points and the Lemma**: If μ is a nonzero extreme point of $U(E)$ and g is a bounded Borel function that integrates to zero with every function in E, then g must be almost everywhere constant with respect to μ.
- **Contradiction and Conclusion**: By continuity and the separation of points condition, it is shown that such a μ leads to a contradiction unless E meets the density condition.

Appendix B

Gini Coefficient

The Gini Coefficient is a measure of statistical dispersion intended to represent the inequality of a distribution, often used in machine learning to evaluate the performance of classification models.

The Gini Coefficient, G, is defined as:

$$G = 1 - \sum_{i=1}^{n} p_i^2$$

where p_i is the probability of an item being classified into the i-th category and n is the number of categories. It ranges between 0 (perfect equality) and 1 (maximal inequality) [51].

Application in Machine Learning

In machine learning, particularly in the context of classification and decision trees, the Gini Coefficient is used to measure the "impurity" of a dataset. It helps in making decisions about splitting nodes in decision trees by quantifying how mixed the classes are in a dataset.

- **Node Impurity**: For a given node, the Gini Coefficient calculates the likelihood of incorrect classification of a randomly chosen element if it were randomly labeled according to the distribution of labels in the node [52].
- **Splitting Criterion**: During the construction of a decision tree, the Gini Coefficient is used to evaluate splits. A split that results in lower Gini impurity is preferred, as it indicates a purer (less mixed) node [51].

Calculation Example

Consider a binary classification problem with two classes, A and B. If a node contains 10 instances of class A and 20 instances of class B, the Gini Coefficient for the node is calculated as follows:

$$p_A = \frac{10}{30} = \frac{1}{3}, \quad p_B = \frac{20}{30} = \frac{2}{3}$$

$$G = 1 - \left(\left(\frac{1}{3}\right)^2 + \left(\frac{2}{3}\right)^2\right) = 1 - \left(\frac{1}{9} + \frac{4}{9}\right) = 1 - \frac{5}{9} = \frac{4}{9}$$