Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Πληροφορικης και Υπολογιστων

Εργαστηριο Συστηματων Τεχνητης Νοημοσυνης και Μαθησης

Actionable Recourse Summaries via Optimal Decision Trees

Diploma Thesis

by

Ioannis Konstantinos Chatzis

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2025

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Πληροφορικής και Υπολογιστών
Εργαστήριο Συστημάτων Τεχνητής Νοημοσύνης και Μάθησης

Actionable Recourse Summaries via Optimal Decision Trees

Diploma Thesis

by

Ioannis Konstantinos Chatzis

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 11η Νοεμβρίου, 2025.

.....	.....	.....
Γεώργιος Στάμου	Αθανάσιος Βουλόδημος	Α.-Γ. Σταφυλοπάτης
Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.	Ομότιμος Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2025

.....

Ιωαννης Κωνσταντινος Χατζης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Το πρόβλημα του *Actionable Recourse* στοχεύει στην εύρεση εφικτών και χαμηλού κόστους αλλαγών στα χαρακτηριστικά εισόδου, οι οποίες ανατρέπουν τις αποφάσεις μη επεξηγήσιμων μοντέλων προβλέψεων, ώστε να επιτευχθεί μία διαφορετική και επιθυμητή πρόβλεψη. Οι περισσότερες εφαρμογές επεξηγήσεων με αντιπαραδείγματα βρίσκουν βέλτιστες τροποποιήσεις, ξεχωριστά για κάθε είσοδο. Ωστόσο, αρκετές εφαρμογές χρειάζονται διαφάνεια και συνέπεια στον τρόπο επιλογής των αλλαγών για την ταυτόχρονη επεξήγηση πληθυσμών. Σχετικές μέθοδοι που εφαρμόζουν επεξηγήσεις σε ομάδες δειγμάτων εισόδου, πετυχαίνουν ήδη την ανάθεση αλλαγών σε επεξηγήσιμες περιοχές του χώρου εισόδου, ωστόσο, δεν έχει υπάρξει κάποια δουλειά που να χρησιμοποιεί βέλτιστα δυαδικά δέντρα αποφάσεων, τα οποία αποδεδειγμένα εγγυόνται καθολικά βέλτιστες λύσεις. Στην εργασία αυτή, υιοθετούμε την μέθοδο των βέλτιστων δέντρων αποφάσεων, βάσει της υλοποίησης μέσων δυναμικού προγραμματισμού, για να εξετάσουμε το πρόβλημα μέσω μίας διαχωρίσιμης και καθολικά βέλτιστης διατύπωσης.

Παρουσιάζουμε το *SOGAR (Summaries of Optimal and Global Actionable Recourse)*, που αποτελεί μια μέθοδο υλοποίησης ενός δι-αντικειμενικού προβλήματος βελτιστοποίησης βασισμένο στο *STreeD (Separable Trees with Dynamic Programming)*. Υπό τις υποθέσεις ότι οι είσοδοι της μεθόδου είναι δυαδικής μορφής (binarized) και της διαχωρισιμότητας (separability), το SOGAR κατασκευάζει ένα σύνολο Pareto, το οποίο περιλαμβάνει καθολικά μη-κυριαρχούμενες (non-dominated) βέλτιστες λύσεις υπό την μορφή δεντρικών δομών, έτσι ώστε να ελαχιστοποιείται το κόστος προσπάθειας κάθε τροποποίησης και να μεγιστοποιείται η επιδραστικότητα της. Σε κάθε κόμβο φύλλο των δέντρων, ανατίθεται μία βέλτιστη τροποποίηση, και οι συναρτήσεις στόχοι που βελτιστοποιούν αυτήν την τροποποίηση λειτουργούν με πλήρως συμβατό τρόπο με τις μεθόδους δυναμικού προγραμματισμού που λειτουργούν βάσει αναδρομικών σχέσεων. Το αποτέλεσμα, είναι ένα σύνολο από μη κυριαρχούμενες που είναι εξίσου βέλτιστες και δίνουν στον χρήστη του συστήματος, την δυνατότητα επιλογής λύσεων βάσει κριτηρίων προσαρμοσμένων του εκάστοτε πεδίου ορισμού του προβλήματος.

Εξετάζουμε την επίδοση της μεθόδου SOGAR, σε τέσσερα σύνολα δεδομένων πινακοειδούς μορφής, και παρατηρήσαμε βελτιωμένη επίδοση και γενίκευση, συγκριτικά με σχετικές μεθόδους του πεδίου. Συνολικά, η μέθοδος, αναδεικνύει την επίδραση των βέλτιστων δέντρων αποφάσεων, στην εύρεση καθολικά βέλτιστων λύσεων και συνοψίσεων Actionable Recourse, οι οποίες εξίσου επιφέρουν επεξηγησιμότητα και βελτίωση της επίδοσης βάσει μετρικών σύγκρισης. Ταυτόχρονα, η μέθοδος αναπαριστά μία διατύπωση που μπορεί να αναπαραχθεί και να επεκταθεί σε μελλοντικές έρευνες.

Λέξεις-Κλειδιά — Επεξηγήσεις με Αντιπαραδείγματα, Καθολικές Επεξηγήσεις με Αντιπαραδείγματα, Ανακατεύθυνση Προβλέψεων, Συνοψίσεις Ανακατεύθυνσης Προβλέψεων, Βέλτιστα Δέντρα Αποφάσεων, Επεξηγησιμότητα, Επεξηγήσιμη Τεχνητή Νοημοσύνη.

Abstract

Actionable Recourse aims to return feasible, low-cost edits on input features that flip black-box predictive models' decisions to a desired outcome. While most counterfactual explanation methods optimize actions per instance, many applications require transparent, consistent prescriptions to populations of affected individuals. Other group-level recourse methods already handle the setting by assigning actions to interpretable regions, but their optimization is not inherently aligned with modern dynamic programming tree learners, which are proven to guarantee global optimum. Here, we adapt *optimal decision trees* to examine this case under a separable, globally optimal formulation.

We introduce *SOGAR* (Summaries of Optimal and Global Actionable Recourse), a bi-objective optimization task built on STreeD (separable trees with dynamic programming). Under standard binarization and separability assumptions, SOGAR constructs a Pareto front of globally optimal trees that jointly minimize cost and maximize the effectiveness of edits. Each leaf is assigned an optimal action, and the objectives are element-wise additive, enabling dynamic programming recurrences. The outcome is a set of non-dominated, interpretable policies which are represented by different trees with near-tied scores, so users can select by domain preference rather than retrain models.

Across four tabular datasets, SOGAR demonstrates a competitive performance relative to related work, and stable generalization for compact trees. Overall, SOGAR shows that globally optimal tree structures can deliver group-level recourse summaries that are both interpretable and quantitatively strong. At the same time, it delivers a formulation that is reproducible and extendable to future tasks.

Keywords — Counterfactual Explanations, Global Counterfactual Explanations, Actionable Recourse, Actionable Recourse Summaries, Optimal Decision Trees, Explainability, XAI.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου κ. Γιώργο Στάμου και τον κ. Θάνο Βουλόδημο για την εμπιστοσύνη που μου έδειξαν, δίνοντάς μου τη δυνατότητα να εκπονήσω τη διπλωματική μου εργασία στο Εργαστήριο Συστημάτων Τεχνητής Νοημοσύνης και Μάθησης, καθώς και για την καθοδήγησή τους σε όλη τη διάρκεια αυτής της διαδικασίας.

Εξίσου, εκφράζω την ειλικρινή μου ευγνωμοσύνη προς τον συνεπιβλέποντα, υποψήφιο διδάκτορα Ιάσονα Λιάρτη, για την καθοδήγηση, τις δημιουργικές και ερευνητικά εύστοχες ιδέες που έδωσαν κατεύθυνση και βάθος στην εργασία, αλλά και για τη συνεπή και άψογη συνεργασία καθ' όλη τη διάρκειά της.

Τέλος, ευχαριστώ τους γονείς μου, που με στηρίζουν με απόλυτη ανιδιοτέλεια και αυταπάρνηση, σε κάθε μου βήμα κι επιλογή, και τους φίλους μου με τους οποίους μπορούσα να μοιραστώ το συναισθηματικό φορτίο και κάθε μικρή νίκη της πορείας, κάνοντας την πιο εύκολη και δημιουργική, και αφήνοντάς μου μία βαθιά ριζωμένη ανάμνηση ανεκτίμητης πνευματικής αξίας.

Γιάννης Χατζής, Νοέμβριος 2025

Contents

List of Figures

List of Tables

Chapter 0

Εκτεταμένη Περίληψη στα Ελληνικά

Η Τεχνητή Νοημοσύνη (AI), πλήρως ενταγμένη στην δομή της κοινωνίας, πλέον επηρεάζει κρίσιμα κοινωνικοοικονομικά πεδία, όπως η πίστωση, εργασία, εκπαίδευση και η υγεία, διαμορφώνοντας αποφάσεις που σε ορισμένες περιπτώσεις επιφέρουν επιπτώσεις στην έκβαση σημαντικών διαδικασιών στις οποίες τον μοναδικό όπου κατά κανόνα είχε ο άνθρωπος. Παρά την υψηλή ακρίβεια των σύγχρονων μοντέλων προβλέψεων, η λειτουργική τους αδιαφάνεια για τους άμεσα επηρεαζόμενους εγείρει ζητήματα μεροληψίας του συστήματος, ιδίως όταν μικρές μεταβολές στα δεδομένα εισόδου μπορούν να ανακατευθύνουν ριζικά την σταδιοδρομία ατόμων.

Η κλασική απάντηση σε αυτό το πρόβλημα είναι οι *επεξηγήσεις με αντιπαραδείγματα σε ατομικό επίπεδο εισόδου* (instance-level counterfactuals), οι οποίες υποδεικνύουν τις ελάχιστες τοπικές μεταβολές που αναστρέφουν την απόφαση ενός μοντέλου. Αν και χρήσιμες, οι μεμονωμένες επεξηγήσεις συχνά υστερούν σε καθολική συνέπεια και μπορούν να παράγουν αντιφατικές προτάσεις για παρεμφερή χαρακτηριστικά σε ξεχωριστές εισόδους, κάνοντας έτσι, πιο δύσκολο τον έλεγχο μεροληψίας σε κλίμακα που εξετάζονται πληθυσμιακές ομάδες έναντι μεμονωμένων περιπτώσεων. Ως εναλλακτική, τα Actionable Recourse Summaries, στοχεύουν σε συνεκτικές, δομημένες οδηγίες για ομάδες ατόμων με παρεμφερή χαρακτηριστικά. Τα *Δέντρα Απόφασης (DT)*, χάρη στην ερμηνευσιμότητά τους, αποτελούν φυσική πλατφόρμα για την παραγωγή τέτοιων συνοψίσεων.

Μία σημαντική σχετική έρευνα είναι τα *Counterfactual Explanation Trees (CET)* [1], όπου τροποποιήσεις των χαρακτηριστικών ανατίθενται στα φύλλα ενός δέντρου. Παρά τη συμβολή τους προς την κατεύθυνση της καθολικής παροχής recourse, τα CET στηρίζονται σε ευριστικές, τοπικά βέλτιστες βελτιστοποιήσεις ανά φύλλο, χωρίς να ελέγχουν εάν η δεντρική δομή είναι καθολικά βέλτιστη. Αυτό συνεπάγεται, ότι η διατύπωση δεν εγγυάται βέλτιστες συνολικές επιλογές ενεργειών, αφήνοντας ανοιχτό το ερώτημα κατά πόσο μια διατύπωση με την βοήθεια Βέλτιστων Δέντρων Αποφάσεων (*Optimal Decision Trees (ODT)*) μπορεί να αποδώσει βεβαιότητα των λύσεων με αυστηρότερη συνέπεια.

Η παρούσα εργασία προτείνει μία διαφανή μέθοδο *Actionable Recourse Summaries* βάσει βέλτιστων δεντρικών δομών. Αξιοποιούμε το *Separable Trees with Dynamic Programming (STreeD)* [2], το οποίο παρέχει καθολική βεβαιότητα βελτιστοποίησης υπό τις συνθήκες διαχωρισιμότητας. Η σχεδίαση της μεθόδου γίνεται ως δι-αντικειμενικό πρόβλημα. Αποτελείται τόσο από την ελαχιστοποίηση του *κόστους ενεργειών*, όσο και την ταυτόχρονη μεγιστοποίηση των *επιτυχιών αναστροφής* (flip rate) όσον αφορά την πρόβλεψη του μοντέλου. Βάσει τροποποίησης των δεδομένων εισόδου σε πλήρως δυαδική μορφή και τα κριτήρια διαχωρισιμότητας, οι στόχοι είναι *προσθετικοί ανά στοιχείο*, επιτρέποντας την υλοποίηση σε δυναμικό προγραμματισμό και την κατασκευή του συνόλου *Pareto* μη-κυριαρχούμενων λύσεων, ώστε ο χρήστης να επιλέγει λύσεις με διαφάνεια όσον αφορά την αλληλοεπηρεαζόμενη σχέση των στόχων του κόστους και της αποτελεσματικότητας, χωρίς την ανάγκη επανεκπαίδευσης του υποκείμενου μοντέλου.

Συνεισφορές.

- Συστηματοποιούμε τη σχέση των επεξηγήσεων με αντιπαραδείγματα με το actionable recourse στο πλαίσιο της επεξηγήσιμης και υπεύθυνης μηχανικής μάθησης.
- Εισάγουμε την νέα διατύπωση της μεθόδου του SOGAR βάσει της υποδομής του STreeD για *βέλτιστα δέντρα* που λύνουν το πρόβλημα του Actionable Recourse, που αναθέτουν συνεπείς και χαμηλού κόστους

ενέργειες σε ομάδες δειγμάτων εισόδου με καθολική βελτιστότητα που αποδεικνύεται ρητά.

- Τέλος, τεκμηριώνουμε εμπειρικά, σε πρότυπα πινακοειδή σύνολα δεδομένων, βελτιώσεις έναντι των σχετικών ερευνητικών εργασιών, τόσο στην σύγκριση κόστους και του flip-rate, με διαφανή γενίκευση για συμπαγείς δομές δέντρων.

Contents

0.1 Θεωρητικό υπόβαθρο

Η ενότητα αυτή συνοψίζει το αναγκαίο γνωσιακό υπόβαθρο Μάθησης Μηχανής, προκειμένου να διευκολύνει την κατανόηση των βασικών εννοιών που αξιοποιούνται στη διπλωματική εργασία. Στόχος είναι η συνοπτική αλλά και ουσιαστική θεμελίωση των αρχών πάνω στις οποίες στηρίζεται η μεθοδολογία που προτείνουμε, καθώς και η ανάδειξη της σχέσης μεταξύ βελτιστοποίησης, ερμηνευσιμότητας και επεξηγησιμότητας στην λήψη αποφάσεων.

0.1.1 Μάθηση Δέντρων Αποφάσεων

Ορισμός προβλήματος

Εξετάζοντας το πρόβλημα της εύρεσης δέντρων αποφάσεων βάσει της προσέγγισης της Μηχανικής Μάθησης με επίβλεψη (ενώ υπάρχουν και μη-επιβλεπόμενες υλοποιήσεις), θεωρούμε ένα δεδομένο σύνολο εκπαίδευσης $D = \{(x_i, y_i)\}_{i=1}^n$ με $x_i \in X \subset \mathbb{R}^D$ και $y_i \in Y$, και αναζητούμε ένα μοντέλο $f \in \mathcal{F}$ που ελαχιστοποιεί το εμπειρικό σφάλμα με συντελεστή κανονικοποίησης ως προς την πολυπλοκότητα:

$$\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell\big(f(x_i), y_i\big) \; + \; \lambda\, \Omega(f).$$

Στα δέντρα αποφάσεων, το μοντέλο f είναι μια ιεραρχική δομή T που χωρίζει αναδρομικά τον χώρο χαρακτηριστικών X σε μη επικαλυπτόμενους υποχώρους R_ℓ και αναθέτει στους τελικούς κόμβους που είναι τα φύλλα της δομής, σταθερές προβλέψεις c_ℓ:

$$f_T(x) = \sum_{\ell=1}^{L} c_\ell \, \mathbf{1}_{\{x \in R_\ell\}}, \qquad \min_{T \in \mathcal{T}} \frac{1}{n} \sum_{i=1}^{n} \ell\big(f_T(x_i), y_i\big) + \alpha |T|.$$

Πώς λειτουργεί ένα δυαδικό δέντρο αποφάσεων

Κάθε εσωτερικός κόμβος εφαρμόζει μια σχέση κατωφλίου σε ένα χαρακτηριστικό (σε σύστημα συντεταγμένων βασισμένο σε άξονες των χαρακτηριστικών εισόδου) και χωρίζει τον πληθυσμό δειγμάτων σε δύο υποσύνολα (*αριστερά και δεξιά*). Η διαδικασία επαναλαμβάνεται μέχρι να ικανοποιηθεί το προκαθορισμένο κριτήριο τερματισμού.

Πρόβλεψη φύλλου: για την διαδικασία παλινδρόμησης $c_\ell = \frac{1}{|R_\ell|} \sum_{x_i \in R_\ell} y_i$, ενώ για την ταξινόμηση $c_\ell = \arg\max_k p_k(R_\ell)$, όπου p_k είναι η εμπειρική συχνότητα της κλάσης k στο φύλλο.

Κριτήριο διάσπασης (impurity reduction)

Σε κόμβο t με δείγματα S_t, επιλέγουμε μια διάσπαση (split) που μεγιστοποιεί τη μείωση του μέτρου του impurity:

$$\Delta I \; = \; I(t) \; - \sum_{j \in \{\mathrm{L},\mathrm{R}\}} \frac{N_j}{N_t} \, I(j),$$

όπου N_t ο αριθμός δειγμάτων στον κόμβο γονέα και N_j στα παιδιά. Τυπικές μετρικές impurity είναι οι:

$$\text{Gini: } I(t) = \sum_k p_k(1 - p_k), \qquad \text{Entropy: } I(t) = -\sum_k p_k \log p_k, \qquad \text{Variance/MSE (regression)}$$

Βασικός άπληστος αλγόριθμος Μάθησης Δέντρων Αποφάσεων (Top-Down Induction)

1. Ξεκίνα από τη ρίζα με όλα τα δεδομένα.
2. Για κάθε χαρακτηριστικό και πιθανό κατώφλι που μπορείς να πραγματοποιήσεις μια διάσπαση του τρέχοντος συνόλου δεδομένων, υπολόγισε ΔI.
3. Εφάρμοσε την διάσπαση (split) που επιφέρει το μέγιστο ΔI.
4. Επανάλαβε αναδρομικά την διαδικασία στα παιδιά.
5. Τερμάτισε τον αλγόριθμο, όταν ικανοποιηθεί κάποιο κριτήριο τερματισμού, όπως, η προσπέλαση του μέγιστου βάθους, ή η μηδενική ή πολύ μικρή τιμή του ΔI κ.λπ.

Έλεγχος πολυπλοκότητας

Τα δέντρα τείνουν να υπερπροσαρμόζουν, λόγω μεγάλου βάθους ή μεγάλο αριθμό φύλλων, άρα απαιτείται η ρύθμιση της πολυπλοκότητας για καλή γενίκευση της διαδικασίας απόφασης. *Πρόωρο pruning* (όρια βάθους, ελάχιστα δείγματα, ελάχιστο κέρδος πληροφορίας) σταματά νωρίς την ανάπτυξη του δέντρου ώστε να περιορίσει τη διακύμανση η οποία επιφέρει ρίσκο υποπροσαρμογής στα δεδομένα εκπαίδευσης. *pruning με καθυστέρηση* (π.χ. cost–complexity) πρώτα αναπτύσσει το πλήρες δέντρο και έπειτα ελαχιστοποιεί

$$\mathrm{Error}(T) + \lambda\, \Omega(T),$$

με επιλογή λ μέσω cross-validation [3, 4]. Έτσι επιτυγχάνεται ρύθμιση της μεροληψίας και διακύμανσης του μοντέλου, κάτι που αναδεικνύει ότι τα πιο απλά δέντρα είναι σταθερότερα κι ερμηνεύσιμα. Σύγχρονες προσεγγίσεις επεκτείνουν την ποινή πέρα από το $|T|$ όπως [5, 6].

0.1.2 Βέλτιστα Δέντρα Αποφάσεων

Η μάθηση *βέλτιστων* δέντρων αποφάσεων είναι NP-πλήρες πρόβλημα όσον αφορά την απόφαση και NP-δύσκολο όσον αφορά την εύρεση της βέλτιστης δεντρικής δομής [7]. Τυπικά, με ένα δεδομένο σύνολο εκπαίδευσης $D = \{(x_i, y_i)\}_{i=1}^n$ και συνάρτηση κόστους $L(T)$, συνήθως εκφράζει το σφάλμα ή τον αριθμό εσωτερικών κόμβων, αναζητούμε την ελαχιστοποίηση:

$$\min_{T \in \mathcal{T}} L(T) \quad \text{s.t} \quad f_T(x_i) = y_i,\ i = 1, \ldots, n,$$

η οποία επιστρέφει πλήθος πιθανών δεντρικών δομών που αυξάνει **υπερεκθετικά** με τον αριθμό χαρακτηριστικών p και το βάθος d, $|\mathcal{T}(p,d)| = O\big((2p)^{2^d-1}\big)$, καθιστώντας την εξαντλητική αναζήτηση ανέφικτη ακόμη και για μέτριες κλίμακες.

Οι κλασικοί αλγόριθμοι (CART/ID3) χρησιμοποιούν άπληστες, τοπικές επιλογές διαχωρισμών. Βάσει της ερευνητικής άνθισης τόσο σε θεωρητικό (αλγοριθμικό) όσο και σε μηχανικό (hardware), ενισχύθηκαν οι *ακριβείς* μέθοδοι που αναζητούν καθολικά βέλτιστες δομές. Από αυτές εξετάζουμε τρεις βασικές κατηγορίες προσέγγισης του προβλήματος βέλτιστων δέντρων αποφάσεων:

1. Μικτού ακέραιου προγραμματισμού (MIO),
2. Περιορισμών/Ικανοποιησιμότητας Λογικών Τύπων (CP/SAT) και
3. **Δυναμικού προγραμματισμού** (DP).

Οι δύο πρώτες κωδικοποιούν το πρόβλημα σε ενιαία αναζήτηση με μείωση του χώρου αναζήτησης μέσω pruning, ενώ ο δυναμικός προγραμματισμός εκμεταλλεύεται *το κριτήριο της βελτιστότητας* (Bellman) και *διαχωρισιμότητα* για να συνθέσει το καθολικό optimum από βέλτιστα υποδέντρα.

MIO: ενιαία μαθηματική διατύπωση

Οι μέθοδοι OCT/BinOCT/FlowOCT [5, 8, 9] μοντελοποιούν ταυτόχρονα την δομή του δέντρου, την δρομολόγηση των δειγμάτων εισαγωγής του αλγορίθμου και τις ετικέτες των φύλλων με δυαδικές/συνεχείς μεταβλητές και περιορισμούς που επιβάλουν συνέπεια στις συγκρίσεις και τις αναθέσεις. Το κλασικό OCT ελαχιστοποιεί την σχέση

$$\frac{1}{n} \sum_{i,\ell,k} \mathbb{1}\{y_i \neq k\}\, z_{i\ell} c_{\ell k} + \alpha \sum_{\ell} L_\ell,$$

με **Branch and Bound** και (συχνά) με γραμμικοποίηση των μη γραμμικών όρων. Το BinOCT μειώνει τους περιορισμούς από $O(n2^D)$ σε $O(2^D)$ κωδικοποιώντας δυαδικά τα μονοπάτια, ενώ το FlowOCT διατυπώνει τη δρομολόγηση των δεδομένων ως ροή δικτύου με πιο περιορισμένες (tight) χαλαρώσεις στις μεταβλητές περιορισμού. Τα MIO επιτυγχάνουν καθολικά βέλτιστα σε μέτρια βάθη/μεγέθη, αλλά το κόστος αυξάνει έντονα με την αύξηση του βάθους d, των κατωφλίων και τα συνεχή χαρακτηριστικά των δειγμάτων εισόδου.

CP/SAT: αναζήτηση με λογικές μεταβλητές

Οι SAT/CP διατυπώσεις [8, 10, 11] εκφράζουν τη δομή του δέντρου ως λογικούς τύπους (CNF) ή υψηλού επιπέδου περιορισμούς (AtMostOne, reified ανισότητες). Η επίλυση βασίζεται στην μάθηση λογικών όρων/προτάσεων και διακλαδώσεων. Είναι ιδιαίτερα αποδοτικές σε διακριτά δεδομένα και μικρά βάθη ($D \leq 5$), αλλά κλιμακώνονται δύσκολα με την αύξηση του πλήθους κατωφλίων ή συνεχείς μεταβλητές.

Δυναμικός προγραμματισμός

Ο **δυναμικός προγραμματισμός** αντιμετωπίζει το βέλτιστο δέντρο ως σύνθεση *βέλτιστων υποδέντρων*. Για υποσύνολο δειγμάτων $S \subseteq D$ σε κόμβο με υπόλοιπο βάθος d, η τιμή κόστους $C(S, d)$ ικανοποιεί την αναδρομή:

$$C(S,d) = \begin{cases} \min\limits_{y \in Y} \mathrm{Err}(S, y), & d = 0, \\ \min\limits_{j,\theta} \big\{ C(S_L(j,\theta), d-1) + C(S_R(j,\theta), d-1) \big\}, & d > 0, \end{cases}$$

δηλαδή *το κόστος του γονέα είναι το άθροισμα* των βέλτιστων παιδιών για κάθε πιθανό split. Η αρχή της βελτιστότητας διασφαλίζει ότι κάθε βέλτιστο δέντρο περιέχει βέλτιστα υποδέντρα.

Πρακτικές υλοποιήσεις δυναμικού προγραμματισμού Πρώιμες μέθοδοι (DL8) [12] αποθήκευσαν σε cache βέλτιστες λύσεις ανά υποπρόβλημα, δηλαδή ανά υποσύνολο δειγμάτων, ώστε να αποφεύγονται επαναλήψεις των λύσεων ταυτόσημων καταστάσεων. Το DL8.5 [13] πρόσθεσε δυναμικά *άνω και κάτω φράγματα* για άμεσο κλάδεμα, επιτυγχάνοντας τάξεις μεγέθους ταχύτερη επίλυση σε πλήρως δυαδικά δεδομένα. Το *MurTree* [14] εισήγαγε αναπαραστάσεις bitset για ταχύτατες πράξεις σε υποσύνολα και *similarity bounds* για πρόωρο αποκλεισμό, κλιμακώνοντας σε δεκάδες χιλιάδες δείγματα με διατήρηση παγκόσμιας βελτιστότητας. Άλλες, υβριδικές προσεγγίσεις, όπως π.χ. το Branches [15] με AND/OR γράφους και AO*, συνδυάζουν ευριστικές κατευθύνσεις αναζήτησης με cache δυναμικού προγραμματισμού, παρέχοντας την εγγύηση βέλτιστου τερματισμού.

Γιατί επιλέχθηκε η προσέγγιση δυναμικού προγραμματισμού;

- **Καθολική βελτιστοποίηση με τοπική συνέργεια:** η αναδρομή επιτρέπει *σύνθεση* λύσεων χωρίς να πλοηγούμαστε ρητά σε όλο τον εκθετικό χώρο δομών.
- **Pruning μέσω φραγμάτων:** κάτω και άνω φράγματα και *memoization* εξαφανίζουν τεράστιες περιοχές αναζήτησης, και τα ταυτόσημα states λύνονται μία φορά.
- **Ευελιξία αξιολόγησης συνάρτησης βελτιστοποίησης:** Η μέθοδος του STreeD [2] δίνει την δυνατότητα επιλογής περισσότερων κριτηρίων αξιολόγησης ταυτόχρονα, μέσω δι-αντικειμενικής βελτιστοποίησης με λύσεις που επιστρέφουν σύνολα Pareto, με πολλαπλές εξίσου βέλτιστες λύσεις.

Σύνοψη Ο δυναμικός προγραμματισμός έχει εξελιχθεί από μικρές κλίμακες σε πρακτικές, ακριβείς λύσεις για χιλιάδες έως εκατοντάδες χιλιάδες δείγματα, παρέχοντας διαφάνεια και εγγυήσεις βελτιστότητας που λείπουν από τις άπληστες μεθόδους. Σε αντίθεση με τις προσεγγίσεις των MIO/CP/SAT, που στηρίζονται σε ενιαία καθολική αναζήτηση, ο δυναμικός προγραμματισμός έχει ένα προβάδισμα λόγω της *επαναχρησιμοποίησης υπολύσεων* και της *προοδευτικής σύνθεσης* υπολύσεων, καθιστώντας εφικτή την ακριβή βελτιστοποίηση δέντρων υπό σύνθετα κριτήρια.

0.1.3 Επεξηγησιμότητα

Βασικοί όροι

- **Ερμηνευσιμότητα (interpretability):** βαθμός στον οποίο ο άνθρωπος κατανοεί τη συμπεριφορά ενός μοντέλου f, κρίνεται από την δομή του μοντέλου, την διαφάνεια των εσωτερικών του κανόνων και την πολυπλοκότητα του.
- **Επεξηγησιμότητα (explainability):** αναφέρεται σε μηχανισμούς E που παράγουν εξηγήσεις, με τέτοιο τρόπο ώστε το μοντέλο να μπορεί να είναι ερμηνεύσιμο, όπως π.χ. post-hoc μηχανισμοί που εξετάζουν τα αποτελέσματα ενός μοντέλου για να αντιληφθούν μέσα από διαφορετική είσοδο, πόσο αλλάζουν αυτά.

Οι επεξηγήσεις κατηγοριοποιούνται σε δύο βασικά επίπεδα:

- **Τοπικό:** εξηγεί την συμπεριφορά του μοντέλου σε γειτονικές περιοχές Π_x μέσω τοπικών δειγμάτων g_x ενός πληθυσμού.
- **Καθολικό:** συνοψίζει το μοντέλο f πάνω στο πεδίο ορισμού των χαρακτηριστικών όλων των δειγμάτων εισόδου D με ένα ερμηνεύσιμο g ή μία δομημένη λογική σύνοψη που συμπεριλαμβάνει συνολικές επεξηγήσεις κατά υποομάδες του πληθυσμού.

Άξονες αξιολόγησης

1. **Πιστότητα (F):** $F = 1 - \mathbb{E}_{z\sim\Pi_x}\big[\ell\big(f(z), \varphi(E(x,f),z)\big)\big]$.
2. **Σταθερότητα (S):** μικρές μεταβολές $x' \approx x$ δεν αλλάζουν ουσιωδώς το $E(x', f)$.
3. **Κατανοησιμότητα/Πολυπλοκότητα (I):** μήκος περιγραφής $L(e)$ (κανόνες, βάθος),

όπου το E αποτελεί τον μηχανισμό επεξήγησης.

Μεθοδικές οικογένειες (ενδεικτικά)

- **Αποδόσεις χαρακτηριστικών:** τα LIME/SHAP[16, 17], κάνουν εξέταση σε kernel, υπόβαθρο, αλληλεπιδράσεις.
- **Πρωτότυπα/παραδείγματα:** αντιπροσωπευτικότητα, ποικιλία, κάλυψη ορίου απόφασης [18].
- **Έννοιες/αναπαραστάσεις:** ευθυγράμμιση με ανθρώπινες έννοιες και έλεγχοι εγκυρότητας.
- **Παγκόσμια υποκατάστατα:** ισορροπία πιστότητας–απλότητας και ρητή περιοχή ισχύος.

Σημείωση Διακρίνουμε *συσχετιστικές* από *αιτιακές* εξηγήσεις· οι πρώτες περιγράφουν τα όρια απόφασης του μοντέλου, οι δεύτερες παρεμβάσεις στον μηχανισμό παραγωγής δεδομένων.

Επεξηγήσεις με Αντιπαραδείγματα (Counterfactuals)

ορισμός (σε εκπαιδευμένο μοντέλο f) Δοθέντος στόχου y^*, παραδείγματος x και εφικτότητας $F(x)$:

$$\min_{x'\in F(x)} C(x,x') \;\; \text{s.t} \;\; f(x') = y^*,$$

όπου C ελέγχει *εγγύτητα/σπανιότητα* (π.χ. ℓ_1/ℓ_2, sparsity).

Αιτιακός ορισμός (SCM $\mathcal{M} = \langle U, V, F, P(U)\rangle$)

Με παρεμβάσεις $a \in A(x)$ που σέβονται γράφημα G:

$$\min_{a\in A(x)} \widetilde{C}(a) \;\; \text{s.t} \;\; Y_{A\leftarrow a}(u) = y^*.$$

Ο εφικτός χώρος εισόδων επάγεται ως $F(x) = \{\phi(a; x, \mathcal{M}) : a \in A(x)\}$[19, 20].

Τέσσερις ιδιότητες ποιότητας

- **(proximity) Εγγύτητα** ($d(x,x')$ μικρό),
- **(sparsity) Σπανιότητα** (λίγες συνιστώσες αλλάζουν),
- **plausibility** (on-manifold, υψηλή πυκνότητα),
- **(feasibility) Εφικτότητα** (περιορισμοί στις αλλαγές).

Trade-offs: το sparsity "αντιμάχεται" το plausibility, άρα πρέπει να δηλώνονται ρητά και να ελέγχονται με περιορισμούς ή ποινές [21].

Βασικά μοτίβα βελτιστοποίησης

- **Άμεση αναζήτηση κοντά στο όριο:** ρύθμιση ευριστικών αποστάσεων από νόρμες ℓ_1/ℓ_2 με ρητούς περιορισμούς $F(x)$ [22].
- **Mixed Integer Programming (MIP):** ακριβής εφικτότητα σε μικτά/κατηγορικά δεδομένα, πιστοποιήσεις[23].
- **Plausibility μέσω της γεωμετρίας του χώρου χαρακτηριστικών:** kNN-μονοπάτια υψηλής πυκνότητας ή αναζήτηση σε λανθάνοντα χώρο $x' = G_\theta(z)$ με ελέγχους $F(x)$[24].

Αξιολόγηση και χρήση

Οι τρόποι αξιολόγησης των τεχνικών επεξηγήσεων με αντιπαραδείγματα προκύπτουν άμεσα από την αξιολόγηση των τεσσάρων βασικών ιδιοτήτων που διέπουν την διαδικασία. Πρακτικά εξετάζουμε ποσοστό επιτυχίας (flip) ενός αντιπαραδείγματος όσον αφορά την αλλαγή της ετικέτας από το εκπαιδευμένο μοντέλο, και την ποσοτικοποίηση της εγγύτητας, της σπανιότητας, του plausibility, και της εφικτότητας. Έτσι, τεκμηριώνονται το μοντέλο κόστους, ο ορισμός $F(x)$ και το υπόβαθρο πυκνότητας. Τέλος, θεωρούμε αρκετά σημαντικό να σημειώσουμε ότι τα αλγοριθμικά counterfactuals περιγράφουν το *όριο απόφασης* του f, όχι αναγκαία πραγματικές επιδράσεις, συνεπώς, για πραγματικές παρεμβάσεις απαιτείται αιτιακή (causal) δομή.

0.1.4 Actionable Recourse

Το πρόβλημα του *Actionable Recourse* στοχεύει σε *εφικτές* αλλαγές στα χαρακτηριστικά ενός δείγματος/ατόμου ώστε ένας σταθερός ταξινομητής $f : \mathcal{X} \to \{0, 1\}$ να αλλάξει απόφαση σε $y^\star = 1$ με ελάχιστο κόστος:

$$\min_{a \in A(x)} c(a \mid x) \quad \text{s.t} \quad f(x + a) = y^\star,$$

όπου το σύνολο τροποποιήσεων $A(x)$ κωδικοποιεί τους περιορισμούς των επιτρεπτών τροποποιήσεων και το $c(\cdot)$ το κόστος [23, 25]. Εδώ αναφερόμαστε στην $y^\star$ ως την επιθυμητή ετικέτα που θα επιστρέψει ο ταξινομητής και το δείγμα x πριν την τροποποίηση ήταν ταξινομημένο στην μη επιθυμητή κλάση.

Εφικτότητα & Κόστος Το σύνολο τροποποιήσεων $A(x)$ παραμένει *σταθερό* κατά την διαδικασία εκπαίδευσης και αξιολόγησης, και περιλαμβάνει τις επιτρεπτές αλλαγές βάσει περιορισμούς για μη-μεταβλητά, κατευθυνόμενα και διακριτά χαρακτηριστικά των δειγμάτων εισόδου [22]. Για συγκρισιμότητα, χρησιμοποιούνται συναρτήσεις που λαμβάνουν υπόψιν κόστους (π.χ. σταθμισμένη ℓ_1 με $w_j \propto 1/\text{MAD}$) ή μετρικές που δεν εξαρτώνται από την κλίμακα των χαρακτηριστικών όπως π.χ. η **MPS**:

$$\text{MPS}(a \mid x) = \max_{j \in S(a)} \left| Q_j(x_j + a_j) - Q_j(x_j) \right|,$$

με Q_j την εμπειρική αθροιστική συνάρτηση κατανομής (CDF) του χαρακτηριστικού j [23].

Από τα αντιπαραδείγματα στο recourse Οι επεξηγήσεις με αντιπαραδείγματα δείχνουν «τι θα άλλαζε» αλλά δεν διασφαλίζουν εφικτότητα ή αναγκαστικά ρεαλιστικές αλλαγές, ενώ το actionable recourse επιβάλλει ρητούς περιορισμούς και μετρικές κόστους για την αξιολόγηση των προτεινόμενων αλλαγών [26].

Ακριβές recourse για γραμμικά μοντέλα Για $f(x) = \mathbb{1}\{w^\top x + b \geq 0\}$ προκύπτει η βελτιστοποίηση μέσω MILP:

$$\min_a c(a \mid x) \;\; \text{s.t} \;\; w^\top (x + a) + b \geq 0,$$

επιστρέφοντας καθολική βελτιστοποίηση ή αντίστοιχα την πιστοποίηση ότι το πρόβλημα δεν είναι εφικτό.

Τα *flipsets* (πολλαπλές ελάχιστου κόστους εναλλακτικές με διαφορετική υποστήριξη) δίνουν προκαθορισμένες αλλαγές που εγγυώνται την επιτυχία των προτεινόμενων αλλαγών σε συγκεκριμένους υποχώρους του συνολικού πληθυσμού δειγμάτων [23].

Μη γραμμικά/agnostic Κωδικοποίηση με SMT (Φ_f, Φ_c) επιτρέπει την εξέταση διακριτών μεταβλητών των χαρακτηριστικών και μη διαφορίσιμα κόστη με εγγύηση της ορθότητας σε περιοχές απόφασης [22]. Επιπλέον, μεγάλη απήχηση έχουν οι *path-feasible* (FACE) περιορισμοί σε γράφους γειτνίασης (k-NN), όπου επιβάλλουν παραμονή στο manifold των δεδομένων (ουσιαστικά το πρακτικό πεδίο ορισμού των δεδομένων, όπως έχει παρατηρηθεί από τα δεδομένα εκπαίδευσης) και επιστρέφουν ακολουθίες αλλαγών μικρών βημάτων [27].

Ανθεκτικότητα & αξιολόγηση Απέναντι σε επαναλαμβανόμενες εκπαιδεύσεις του μοντέλου, εκφραζόμενες ως $\{f^{(b)}\}$, μετριούνται ποσοστά επιτυχίας και εφικτότητας, οι κατανομές ελάχιστου κόστους (π.χ. MPS), η σπανιότητα των τροποποιήσεων και η ποικιλία, μέσα από το μέγεθος ή τις επικαλύψεις των flipsets, υπό σταθερά σύνολα $A(x)$ και συναρτήσεις κόστους $c(\cdot)$.

Σύνοψη Το actionable recourse μετατρέπει τις επεξηγήσεις με αντιπαραδείγματα σε ένα πρόβλημα *βελτιστοποίησης υπό περιορισμούς*, παρέχοντας εφικτές, δίκαια κοστολογημένες και κατάλληλες για αναπαραγωγή προτεινόμενες αλλαγής με σαφείς και συνεπείς κανόνες αξιολόγησης.

0.2 Σχετική Έρευνα

0.2.1 Σχετική Έρευνα - Actionable Recourse Summaries

Actionable Recourse Summaries (AReS)

Το AReS συνοψίζει παρεμβάσεις ως μικρό σύνολο κανόνων $R = \{(q, c, c')\}$: αν ένα άτομο ικανοποιεί το *φίλτρο* q και την *τρέχουσα κατάσταση* c, τότε προτείνεται *ενέργεια* c'. Στόχος είναι λίγοι, αναγνώσιμοι κανόνες ($\approx 10 - 15$) που καλύπτουν μεγάλο μέρος του πληθυσμού [28].

Μετρικές

Θεωρούμε X_{aff} τα δείγματα με την ανεπιθύμητη πρόβλεψη από έναν δεδομένο ταξινομητή B. Η μετρική αστοχίας της προτεινόμενης τροποποίησης μετρά πόσες φορές η εφαρμογή της δεν αναστρέφει την ετικέτα:

$$\text{incorrectrecourse}(R) = \sum_{i=1}^{M} \big|\{x \in X_{\text{aff}} :\ x \models q_i \wedge c_i,\ B(\text{substitute}(x, c_i, c_i')) \neq 1\}\big|.$$

Η *κάλυψη* (impact) δείχνει τι ποσοστό του εξεταζόμενου πληθυσμού λαμβάνει *εφαρμόσιμη* πρόταση τροποποίησης (δηλαδή προτείνεται αλλαγή σε τουλάχιστον μία συνιστώσα των χαρακτηριστικών των δειγμάτων):

$$\text{cover}(R) = \big|\{x \in X_{\text{aff}} :\ \exists i\ \ x \models q_i \wedge c_i\}\big|.$$

Κόστη και μέγεθος αλλαγών:

$$\text{featurecost}(R) = \frac{1}{M}\sum_{m=1}^{M} \text{cost}(c_m'), \quad \text{featurechange}(R) = \frac{1}{M}\sum_{m=1}^{M} \text{magnitude}(c_m').$$

Η μετρική $\text{cost}(c_m')$ δεν είναι απαραίτητα κάποια νόρμα ℓ_p που εκφράζει απόσταση, καθώς μπορεί να περιλαμβάνει την έκφραση χρονικής και δομικής δυσκολίας της αλλαγής.

Η μετρική του $\text{magnitude}(c_m')$ είναι το μέγεθος μεταβολής (εκφρασμένο σαν πλάτος) στον χώρο χαρακτηριστικών (πόσες συντεταγμένες αλλάζουν και κατά πόσο).

Ο διαχωρισμός μεταξύ των μετρικός είναι κρίσιμος, αφού η ίδια μεταβολή μεγέθους μπορεί να έχει διαφορετικό πραγματικό κόστος, όπως π.χ. αύξηση 10% μισθού σε άτομα με μεγάλη μισθολογική διαφορά.

Ερμηνευσιμότητα και πολυπλοκότητα

$$\text{size}(R) = |R|, \quad \text{maxwidth}(R) = \max_{(q,c,c')\in R} \#\text{predicates}(q \wedge c), \quad \text{numrsets}(R) = |\text{rset}(R)|.$$

Στόχος: σύντομο κείμενο, περιορισμένη τοπική πολυπλοκότητα, αποφυγή κατακερματισμού κανόνων.

Βελτιστοποίηση και όρια

Ο χώρος αναζήτησης τροποποιείται σε πλήρως διακριτές τιμές (binning) και αναζητούνται μικρά R με χαμηλό κόστος και μέγιστη κάλυψη. Η μέθοδος είναι *προσεγγιστική*, και ο συνδυαστικός χώρος εξερευνάται μερικώς και η κλιμάκωση σε μεγάλα σύνολα είναι απαιτητική. Παρά τα όρια, θέτει χρήσιμα πρότυπα μετρικών και κανόνων για την παραγωγή κατάλληλων συνοψίσεων σε επίπεδο της εξέτασης ενός πληθυσμού.

Counterfactual Explanation Trees (CET)

Το CET εκφράζει τις συνοψίσεις του actionable recourse ως *δέντρα αποφάσεων*,όπου το δέντρο τμηματοποιεί τον χώρο χαρακτηριστικών και κάθε *φύλλο* αναθέτει *μία* τροποποίηση στο υποσύνολο του πληθυσμού που ανήκει στο φύλλο, ισορροπώντας το κόστος και την αποδοτικότητα της αναστροφής της ετικέτας από την μη επιθυμητή στην επιθυμητή [1] .

Αξιολόγηση και μετρικές

Δοθέντος ενός ταξινομητή f και ενός πληθυσμού στόχευσης, ορίζεται η μετρική του *invalidity* ανά δείγμα πληθυσμού ως:

$$\iota_\gamma(a \mid x) = c(a \mid x) + \gamma \cdot \ell_{01}(f(x+a), +1),$$

με $c(a|x)$ να είναι το κόστος της τροποποίησης, το ℓ_{01} η συνάρτηση απώλειας 0–1 (0 αν η ετικέτα αναστρέφεται σε $+1$, αλλιώς επιστρέφει 1) και το $\gamma > 0$ να είναι ένας συντελεστής στάθμισης της συνάρτησης απώλειας (flip-loss).

Ανά φύλλο X:

$$g_\gamma(a \mid X) = \sum_{x \in X} \iota_\gamma(a \mid x), \qquad a_X^\star = \arg\min_{a \in \mathcal{A}(X)} g_\gamma(a \mid X).$$

Μονοτονία ως προς διαμέριση

Για ένα διαμερισμό $X = X_1 \cup X_2$, τέτοιος ώστε $X = X_1 \cap X_2 = \emptyset$ ισχύει:

$$g_\gamma(a_X^\star \mid X) \;\geq\; g_\gamma(a_{X_1}^\star \mid X_1) + g_\gamma(a_{X_2}^\star \mid X_2), \tag{0.2.1}$$

υποδηλώνοντας έτσι, ότι η *λεπτομερέστερη* τμηματοποίηση μπορεί να βελτιώσει την συνολική επίδοση ως προς το αριθμητικό αποτέλεσμα την μετρικής της αξιολόγησης του προβλήματος.

Συνολική συνάρτηση αξιολόγησης δέντρου h:

$$o_{\gamma,\lambda}(h) = \frac{1}{N} \sum_{l \in L(h)} g_\gamma(a_l \mid X_l) + \lambda\, |L(h)|,$$

όπου ο πρώτος όρος είναι η μέση τιμή του invalidity και ο δεύτερος όρος κανονικοποιεί την *πολυπλοκότητα*, βάσει ποινής που επιβάλλεται στον αριθμό φύλλων του δέντρου μέσω λ.

Κόστος MPS

Υιοθετείται η μετρική κόστους *Maximum Percentile Shift* (MPS)[23], για την σύγκριση ετερογενών γνωρισμάτων χωρίς την εξάρτηση από την κλίμακα και το εύρος τιμών κάθε χαρακτηριστικού του χώρου, σύμφωνα με την εμπειρική αθροιστική συνάρτηση κατανομής (CDF) για κάθε χαρακτηριστικό του χώρου. Η MPS εκφράζεται ως

$$c_{\text{MPS}}(x, a) = \max_{j \in A(x)} \Big(Q_j(x'_j) - Q_j(x_j) \Big), \tag{0.2.2}$$

όπου Q_j είναι εμπειρική CDF του χαρακτηριστικού j στον αρχικό χώρο τιμών των χαρακτηριστικών και $x' = x + a$ να είναι το δείγμα που εξετάζεται μετά την εφαρμογή της τροποποίησης a.

Αλγόριθμος Στοχαστικής Τοπικής Αναζήτησης)

Για κάθε επανάληψη του αλγορίθμου ορισμένη από ένα πεπερασμένο T:

1. εισαγωγή/διαγραφή/αντικατάσταση κόμβων με πιθανότητες,
2. αξιολόγηση φύλλων και επιλογή $a^\star$ ανά φύλλο,
3. αποδοχή αλλαγών μόνο αν βελτιώνουν τον στόχο $o_{\gamma,\lambda}$.

Η διαδικασία είναι *προσεγγιστικά βέλτιστη* (χωρίς αυστηρή ολική βέλτιστη λύση), αλλά αποδίδει πρακτικά ισχυρά αποτελέσματα.

Περιορισμοί και πρακτική αξία

Όσον αφορά την κλιμάκωση, όσο μεγαλύτερος είναι χώρος χαρακτηριστικών και το σύνολο τροποποιήσεων, τόσο περισσότερο οδηγείται το μοντέλο σε πιο εκτενέστερη και τουτέστιν πιο δυσχερή επίλυση βελτιστοποίησης MILO ανά φύλλο. Η ποιότητα εξαρτάται από τον αριθμό επαναλήψεων και το action set, για μεγαλύτερα σύνολα δεδομένων. Παρ' όλα αυτά, το CET προσφέρει *διαφάνεια*, μέσω μοναδικών προτάσεων τροποποίησης ανά φύλλο, *σαφή* μετρική αξιολόγησης των λύσεων και *συστηματική* σύνδεση κόστους και επιτυχίας. Συγκριτικά με AReS, η δεντρική δομή της σύνοψης διευκολύνει την ιεραρχική ερμηνεία για την κατανομή και την αιτία της κατανομής των προτεινόμενων τροποποιήσεων, ενώ μοιράζεται τις ίδιες βασικές αρχές μέτρησης αξιολόγησης της αποτελεσματικότητας των λύσεων.

0.2.2 Σχετική Έρευνα - Διαχωρίσιμα Βέλτιστα Δέντρα Αποφάσεων

- Το *Separable Trees with Dynamic Programming (STreeD)* [2] είναι μια μέθοδος βέλτιστων δέντρων που επιλύει διαδικασίες βελτιστοποίησης με Δυναμικό Προγραμματισμό, αξιοποιώντας την ανεξαρτησία υποδέντρων ως ξεχωριστά βέλτιστες λύσεις. Προσφέρει *τάξεις μεγέθους* καλύτερη κλιμάκωση από άλλες μεθόδους σε μέτριας κλίμακας προβλήματα.
- **Δυαδική τροποποίηση χαρακτηριστικών:** Ο χώρος χαρακτηριστικών λαμβάνεται ως δυαδικός ($x_f \in \{0,1\}$). Η *ορθή και πλήρης* διαδικασία τροποποίησης είναι προϋπόθεση για ακριβείς & αποδοτικές αναδρομές.
- **Κλασική DP για σφάλματα ταξινόμησης:** Για μέγιστο βάθος d,

$$T(D,d) = \begin{cases} \min\{|D^+|, |D^-|\}, & d = 0, \\ \min\limits_{f \in F} T(D_f, d-1) + T(D_{\bar{f}}, d-1), & d > 0, \end{cases}$$

  με $D_f = \{(x,k) \in D : x_f = 1\}$. Χρησιμοποιούνται εκτενώς οι τεχνικές δυναμικού προγραμματισμού του memoization και bounds, κάτι που βελτιώνει σημαντικά την ταχύτητα εκτέλεσης, μειώνοντας ταυτόχρονα τον χώρα αναζήτησης και ουσιαστικά αποθηκεύοντας τις μικρότερης τάξης λύσης για απλή ανάκληση από τη μνήμη.
- **Ορισμός διαδικασίας βελτιστοποίησης μέσω του STreeD:** Ορίζεται από την τούπλα

$$o = \langle g, t, \succ, \oplus, c, s_0 \rangle,$$

  όπου g (τοπική αξιολόγηση), t (μετάβαση), $\succ$ (τελεστής σύγκρισης), $\oplus$ (τελεστής συνδυασμού λύσεων), c (περιορισμοί εφικτότητας), s_0 (αρχική κατάσταση).
- **Πολυκριτήρια (Pareto) DP:** Οι κόμβοι επιστρέφουν *σύνολα* μη-κυριαρχούμενων τιμών (αντί μίας) [29].

  Ορισμοί τελεστών για $\Theta \subseteq V$:

  – $\text{feas}(\Theta, s)$. **έλεγχος εφικτότητας**. Κρατά από το σύνολο υποψηφίων τιμών $\Theta \subseteq V$, οι οποίες ικανοποιούν τους περιορισμούς του τρέχοντος state s μέσω του predicate $c(v,s) = 1$. Χρησιμοποιείται για να απορρίπτονται έγκαιρα λύσεις που παραβιάζουν τους περιορισμούς που καθιστούν λύσεις εφικτές π.χ. budgets, ελάχιστα μεγέθη φύλλων, κ.λπ.

– $\mathrm{nondom}(\Theta)$. **Φίλτρο μη-κυριαρχίας**. Επιστρέφει τις μη-κυριαρχούμενες τιμές του Θ, αφαιρώντας κάθε v για το οποίο υπάρχει $v' \in \Theta$ με $v' \succ v$ ως προς τη διάταξη Pareto. Περιορίζει τον μέγεθος του συνόλου Pareto.

– $\mathrm{opt}(\Theta, s) = \mathrm{nondom}(\mathrm{feas}(\Theta, s))$. **Τοπικά βέλτιστο σύνολο**. Συνδυάζει εφικτότητα και μη-κυριαρχία σε ένα βήμα. Είναι ακριβώς το summary που πρέπει να αποθηκεύει κάθε κόμβος για να προχωρήσει η διαδικασία του δυναμικού προγραμματισμού χωρίς απώλεια καθολικής βέλτιστης πληροφορίας.

– $\mathrm{merge}(\Theta_1, \Theta_2, s, f) = \{\, v_1 \oplus v_2 \oplus g(s, f) \,\}$. **Σύνθεση γονέα από παιδιά**. Δημιουργεί όλες τις υποψήφιες λύσεις γονείς, συνδυάζοντας μία τιμή από κάθε παιδί μέσω του τελεστή $\oplus$ και προσθέτοντας (εάν υπάρχει) την συνεισφορά του split f με $g(s, f)$. Απαιτείται η *διατήρηση της διάταξης* του $\oplus$ ώστε η βελτίωση ενός παιδιού να μην χειροτερεύει τον γονέα.

– $T(s, d) = \begin{cases} \mathrm{opt}\{g(s, \hat{k}) : \hat{k} \in K\}, & d = 0, \\ \mathrm{opt}\Big(\bigcup\limits_{f \in F} \mathrm{merge}\big(T(t(s, f, 1), d-1),\, T(t(s, f, 0), d-1),\, s,\, f\big)\Big), & d > 0 \end{cases}$.

Αναδρομή συνόλων.

* Στο βάθος 0 ένα φύλλο επιστρέφει όλες τις εφικτές επιλογές ετικέτας και κρατά τις μη-κυριαρχούμενες.

* Στους εσωτερικούς κόμβους συνδυάζονται οι συναθροίσεις συνόλων Pareto των παιδιών για κάθε πιθανή διάσπαση χαρακτηριστικού f και εφαρμόζεται εκ νέου opt για να αφαιρεθούν μη εφικτές ή κυριαρχούμενες τιμές. Έτσι, η ρίζα επιστρέφει ολόκληρο το σύνολο Pareto για βάθος έως d.

- **Διαχωρισιμότητα (separability):** Κρίσιμη συνθήκη ορθότητας για ένα διαχωρίσιμο πρόβλημα:

$$\mathrm{opt}(\mathrm{merge}(\Theta_1, \Theta_2, s, f), s) = \mathrm{opt}\big(\mathrm{merge}(\mathrm{opt}(\Theta_1, s_1), \mathrm{opt}(\Theta_2, s_2), s, f), s\big),$$

όπου $s_1 = t(s, f, 1)$, $s_2 = t(s, f, 0)$ δύο καταστάσεις παιδιά. Ισχύει όταν:

1. *Markovian* g, t: εξαρτώνται μόνο από (s, a) ή (s, f, δ).
2. **Διατήρηση σειράς διάταξης** (σεβασμός του συδνυαστικού τελεστή $\oplus$ στην σειρά διάταξης που ορίζει ο τελεστής σύγκριση/κυριαρχίας$\succ$) από $\oplus$: αν $v_1 \succ v_1' \Rightarrow v_1 \oplus v_2 \succ v_1' \oplus v_2$.
3. **Αντι-μονοτονική εφικτότητα** c: Αν μια υπολύση δεν μπορεί να είναι εφικτή στα πλαίσια του περιορισμού c τότε καμία λύση μεγαλύτερης τάξης (γονέας) δεν θα μπορεί χρησιμοποιώντας την υπολύση να θεωρηθεί συνολικά εφικτή.

Έτσι η διαχωρισιμότητα, επιτρέπει caching, τοπικό pruning και την ανεξάρτητη επίλυση υποπροβλημάτων.

- **Σύνθεση πολλαπλών στόχων:** Αν δύο διαδικασίες βελτιστοποίησης $o^{(1)}, o^{(2)}$ είναι διαχωρίσιμες στο ίδιο t, ο συνδυασμός $o^{\otimes}$ με $g^{\otimes} = (g^{(1)}, g^{(2)})$, $\succ^{\otimes}$ ο συνολικός τελεστής διάταξης, $\oplus^{\otimes}$ ο συνδυαστικός τελεστής κατά συνιστώσα, και $c^{\otimes} = c^{(1)} \wedge c^{(2)}$ είναι επίσης διαχωρίσιμος. Η διαδικασία επίλυσης του προβλήματος δυναμικού προγραμματισμού επιστρέφει το πολυδιάστατο σύνολο Pareto.

- **Πρακτικές τεχνικές:** Memoization (hashing καταστάσεων), ισχυρά άνω και κάτω φράγματα για πρόωρο pruning, κυριαρχία βάσει παραμέτρων χαλάρωσης ε για έλεγχο πλάτους του συνόλου Pareto, και cache τοπικών τιμών g για $O(1)$ ανακτήσεις προϋπολογισμένων υπολύσεων [14].

- **Πολυπλοκότητα & κλιμάκωση:** Το εύρος του συνόλου Pareto μπορεί να αυξηθεί συνδυαστικά (τυπικό σε ακριβή Pareto enumeration). Τα ελεγχόμενα ε, feasibility thresholds και όρια βάθους/στήριξης κρατούν τη διαδικασία πρακτική.

- **Συμπέρασμα:** Το STreeD δίνει ένα ενοποιημένο, επεκτάσιμο σκελετό για βέλτιστα δέντρα και διαδικασίες βελτιστοποίησης που δεν περιορίζονται σε ένα μόνο στόχο βελτιστοποίησης, με την εγγύηση ορθότητας μέσω της διαχωρισιμότητας και με πρακτικά εργαλεία pruning του χώρου λύσεων. Αυτά αποτελούν καθοριστικά εφόδια για μεθόδους που απαιτούν *καθολικά βέλτιστες*, ερμηνεύσιμες δομές αποφάσεων.

0.3 Προτεινόμενη Μέθοδος

Το **SOGAR** προτείνει μια ενιαία διατύπωση για το πρόβλημα το *Actionable Recourse Summaries*, η οποία επιλύει με *καθολική βελτιστότητα* τη δομή του δέντρου και την ανάθεση τροποποιήσεων ανά φύλλο. Βασισμένο στις αρχές δυναμικού προγραμματισμού και την υποδομή του *STreeD*, το πρόβλημα διατυπώνεται ως ένα διαντικειμενικό, *διαχωρίσιμο* πρόβλημα βελτιστοποίησης, κατά το οποίο ελαχιστοποιούμε το *συνολικό κόστος* των ενεργειών και ελαχιστοποιούμε την *αποτυχία αναστροφής (loss)* της απόφασης του ταξινομητή. Οι λύσεις που προκύπτουν είναι το σύνολο Pareto των μη-κυριαρχούμενων δέντρων. Κάθε φύλλο προδιαγράφει μία ενέργεια που στοχεύει να αντιστρέψει τις αρνητικές αποφάσεις με ελάχιστο κόστος, ενώ ο αλγόριθμος επίλυσης διατηρεί τις βέλτιστες αλληλένδετες αυξομειώσεις κόστους και επιτυχίας σε όλο τον χώρο χαρακτηριστικών. Έτσι, το SOGAR επεκτείνει τον χώρο αναζήτησης πέρα από τοπικές ενέργειες, διασφαλίζει καθολική βελτιστοποίηση υπό σαφείς συνθήκες, και γεφυρώνει την ερμηνευσιμότητα των δέντρων αποφάσεων με τις απαιτήσεις συνέπειας, διαφάνειας και ισοτιμίας των σύγχρονων συστημάτων recourse. Έτσι, παρέχονται διαφανείς και πολιτικές, συνεπείς ανά εξεταζόμενη ομάδα του συνολικού πληθυσμού, για ευαίσθητους τομείς όπως πίστωση, προσλήψεις και υγεία.

0.3.1 Συνεισφορά

Οι συνεισφορές της διπλωματικής εργασίας είναι οι εξής:

- Προτείνουμε την μέθοδο **SOGAR***(Summaries of Optimal and Global Actionable Recourse*, ένα νέο πλαίσιο που επεκτείνει την λογική του Counterfactual Explanations Trees (CET) [1], με την ενσωμάτωση της στην υποδομή της μεθόδου δυναμικού προγραμματισμού του Separable Trees with Dynamic Programming (STreeD) [2]. Μέσω της προσέγγισης αυτής διατηρούμε την ερμηνευσιμότητα των λύσεων και ταυτοχρόνως υπάρχει εγγύηση της καθολικής βελτιστότητας, βάσει όλων των πιθανών βέλτιστων λύσεων που εντάσσονται στο σύνολο Pareto. Το σύνολο αυτό δίνει μία επιπλέον πολυτέλεια στο χρήστη του συστήματος, να έχει επιλογή πολλαπλών λύσεων, ανάλογα με το πεδίο ορισμού και τις ανάγκες τους προβλήματος του.

- Κάθε φύλλο του **δέντρου αποφάσεων** αντιστοιχίζεται σε *μία* εφικτή ενέργεια, η οποία επιλέγεται ώστε να *ελαχιστοποιεί* το συνολικό κόστος μεταβολών χαρακτηριστικών και να *μεγιστοποιεί* τον ρυθμό επιτυχών αναστροφών (*flip rate*) υπό έναν σταθερό *μη ερμηνεύσιμο* ταξινομητή. Η διατύπωση αυτή εξασφαλίζει *καθολικά συνεπείς* και *διαφανείς* προτάσεις.

- Πραγματοποιούμε πειραματική αξιολόγηση σε *πρότυπα πινακοειδή σύνολα* δεδομένων, δείχνοντας ότι το **SOGAR** επιτυγχάνει χαμηλότερο *μέσο κόστος ανατροφοδότησης* και υψηλότερα *flip rates* σε σύγκριση με **CET** και **AReS**, με αποδοτική κλιμάκωση σε μεγαλύτερα βάθη δέντρου και μεγέθη συνόλων.

- Καθιερώνουμε *συστηματικό πλαίσιο* για τη μελέτη της *αλληλένδετης επιρροής* του *κόστους* και της *συνάρτησης απώλειας* μέσω *Pareto* βελτιστοποίησης, προσφέροντας σε πρακτικούς και φορείς ένα *διαφανές εργαλείο* για εφαρμόσιμη και ικανή να αιτιολογηθεί ανατροφοδότηση.

- Η τυπική διαμόρφωση επιτυγχάνεται μέσω της προσαρμογής *του πλαισίου του* ***STreeD***, με τη διατύπωση *διαχωρίσιμης διαδικασίας βελτιστοποίησης* $o = \langle g, t, \oplus, \succ, c, s_0 \rangle$, που πληροί τις *αναγκαίες και ικανές συνθήκες* για την DP αναδρομή με *προϋπολογισμένες* τιμές κόστους και συνάρτησης απώλειας.

0.3.2 Ορισμός Προβλήματος

Ορίζουμε ένα σταθερό *μη ερμηνεύσιμο ταξινομητή* $f : \mathcal{X} \to \{0, 1\}$ και στοχεύουμε την ετικέτα $y^\star = 1$ (ανεπιθύμητη: 0). Θεωρούμε πληθυσμό $D = \{(x_i, f(x_i))\}_{i=1}^n$ και το επηρεαζόμενο υποσύνολο $D_0 = \{x : f(x) = 0\}$. Ορίζουμε χώρο ενεργειών $\mathcal{A}$ και, για κάθε x, εφικτό σύνολο $\mathcal{A}(x) \subseteq \mathcal{A}$ με σπανιότητα έως 3 τροποποιήσεις. Η *συνάρτηση κόστους* $c(a \,|\, x)$ μετρά την προσπάθεια αλλαγών στα γνωρίσματα, ενώ η *συνάρτηση απώλειας (ένδειξη αναστροφής)* είναι $\ell_{01}(f(x + a), 1)$. Το ζητούμενο είναι ένα *Pareto* σύνολο δέντρων βάθους $\leq d$ όπου κάθε φύλλο του **δέντρου αποφάσεων** φέρει μία εφικτή ενέργεια που ελαχιστοποιεί ταυτόχρονα

$$C(\tau) = \sum_{x \in D_0} c\big(a(\ell(x)) \,|\, x\big), \qquad L(\tau) = \sum_{x \in D_0} \ell_{01}\big(f(x + a(\ell(x))), 1\big),$$

δηλαδή

$$\min_{\tau \in \mathcal{T}_d} \big(C(\tau),\, L(\tau)\big),$$

όπου $\mathcal{T}_d$ ο χώρος δυαδικών λύσεων με μέγιστο βάθος d. Για την σύγκριση με άλλες σχετικές μεθόδους, χρησιμοποιούμε και έναν σύνθετο δείκτη που αποκαλούμε *invalidity*, καθότι δεν υπάρχει κάποια άμεση μετάφραση σε ελληνική ορολογία, ο οποίος εκφράζεται ως

$$I(\tau) = \frac{1}{|D_0|} \sum_{x \in D_0} \Big(c\big(a(\ell(x)) \,|\, x\big) + \gamma\, \ell_{01}\big(f(x + a(\ell(x))), 1\big) \Big),$$

με γ ως συντελεστή στάθμισης του κόστους και της συνάρτησης απώλειας. Πρακτικά το invalidity μας επιστρέφει ένα σταθμισμένο άθροισμα το οποίο ρυθμίζει την σημαντικότητα και συνεισφορά της αποδοτικότητας μίας τροποποίησης στην όλη διαδικασίας. Μικρός συντελεστής γ υποδηλώνει μείωση την σημαντικότητας της συνάρτησης απώλειας και μεγαλύτερο γ την αντίστοιχη αύξηση.

Βάσει των παραπάνω παραθέτουμε μια λίστα που περιλαμβάνει κάθε μέρος της θεωρητικής διαδικασίας, για την επίλυση του προβλήματος:

- Σταθερός ταξινομητής $f : \mathcal{X} \to \{0, 1\}$, στόχος $y^\star = 1$.
- Πληθυσμός D και επηρεαζόμενο υποσύνολο $D_0 = \{x : f(x) = 0\}$.
- Εφικτές ενέργειες $\mathcal{A}(x)$ (σπανιότητα ≤ 3).
- Συναρτήσεις: κόστος $c(a \,|\, x)$ και συνάρτηση απώλειας $\ell_{01}(f(x + a), 1)$.
- Pareto στόχος: $\min_{\tau \in \mathcal{T}_d} \big(C(\tau), L(\tau)\big)$ με τ **δέντρο αποφάσεων** .
- Συγκεντρωτικές συναρτήσεις: $C(\tau)$, $L(\tau)$ και δείκτης $I(\tau)$.

0.3.3 Προσαρμογή στο *STreeD* και διαχωρισιμότητα

Ορίζουμε την διαδικασία βελτιστοποίησης κατά τις προδιαγραφές του *STreeD*, δηλαδή ως προς την τούπλα $o = \langle g, t, \oplus, \succ, c, s_0 \rangle$, όπου g είναι η συνάρτηση κόστους και η συνάρτηση απώλειας σε διάνυσμα, t η συνάρτηση μετάβασης κατά διακλάδωση, $\oplus$ ο τελεστής συνδυασμού της συνάρτησης κόστους, $\succ$ ο τελεστής που εκφράζει την Pareto κυριαρχία, c οι περιορισμοί εφικτότητας μιας λύσης και s_0 η αρχική κατάσταση. Μια κατάσταση γράφεται ως $s = \langle X_s, d \rangle$, με υποσύνολο δειγμάτων εισόδου $X_s \subseteq D_0$ και υπόλοιπο βάθος d. Ο χώρος τιμών των λύσεων είναι $V = \mathbb{R}^2_{\geq 0}$, όπου η πρώτη συνιστώσα είναι το αθροιστικό κόστος και η δεύτερη ο αριθμός αποτυχημένων αναστροφών. Ως ετικέτες φύλλων επιτρέπονται όλες οι εφικτές ενέργειες $a \in \mathcal{A}(x)$ (μία ανά φύλλο).

Η συνάρτηση g ορίζεται με προϋπολογισμένες τιμές κόστους και έκβασης ως $g(s, \alpha) = \big(\sum_{x \in X_s} c(\alpha \mid x),\ \sum_{x \in X_s} \ell_{01}(f(x + \alpha), 1)\big)$. Η μετάβαση t για ένα χαρακτηριστικό των δειγμάτων εισόδου $f \in F$ και διακλάδωση $b \in \{0, 1\}$ δίνεται από την σχέση $t(s, f, b) = \langle X_{s,b}, d - 1 \rangle$ με $X_{s,1} = \{x \in X_s : f_{\text{bin}}(x) = 1\}$ και $X_{s,0} = X_s \backslash X_{s,1}$. Ο τελεστής συνδυασμού είναι προκύπτει με άθροιση κατά συνιστώσα $u \oplus v = (u_1 + v_1,\ u_2 + v_2)$, ενώ η κυριαρχία είναι το σύνολο Pareto ορισμένο στο $\mathbb{R}^2_{\geq 0}$ με προαιρετική χαλάρωση ε για αριθμητική ανοχή κατά τις συγκρίσεις. Οι περιορισμοί c επιβάλλουν εφικτότητα των τροποποιήσεων για όλα τα $x \in X_s$, τους οποίους προκαθορίζουμε να είναι το ελάχιστο μέγεθος φύλλου $|X_s| \geq m$ και η σπανιότητα $\|\alpha\|_0 \leq k$ που εκφράζει τον μέγιστο αριθμό των συνολικών αλλαγών ανά τροποποίηση. Η αρχική κατάσταση είναι $s_0 = \langle D_0, d_{\max} \rangle$.

Η διαχωρισιμότητα προκύπτει άμεσα. Πρώτον, g και t είναι Markovian, αφού εξαρτώνται μόνο από την τρέχουσα κατάσταση και την άμεση απόφαση. Δεύτερον, ο τελεστής συνδυασμού $\oplus$ σέβεται τον τελεστή κυριαρχίας $\succ$ ως προς την σειρά που συγκρίνει τις λύσεις μεταξύ τους, επειδή η άθροιση ως προς κάθε στοιχείο στο $\mathbb{R}^2_{\geq 0}$ είναι μονοτονική στο πλαίσιο του συνόλου Pareto. Τρίτον, οι περιορισμοί είναι αντί-μονοτονικοί αφού αν παραβιαστούν σε κάποιο υποδέντρο, παραβιάζονται και σε οποιαδήποτε δεντρική δομή ανώτερης τάξης. Συνεπώς ισχύει η αρχή της βελτιστότητας και η διαδικασία βελτιστοποίησης είναι διαχωρίσιμη υπό τις προϋποθέσεις του *STreeD*.

Η αναζήτηση με δυναμικό προγραμματισμό παράγει το σύνολο Pareto λύσεων $T(s, d)$. Η περιγραφή της αναδρομικής σχέσης που προκύπτει αποτελείται από τα επόμενα δύο σκέλη:

1. Στα φύλλα ($d = 0$) επιστρέφονται οι μη κυριαρχούμενες τιμές $\{g(s, \alpha) : \alpha \in A_{\text{leaf}}(s)\}$.
2. Στους εσωτερικούς κόμβους ($d > 0$) εξετάζονται όλες οι διακλαδώσεις για τα χαρακτηριστικά δειγμάτων εισόδου $f \in F$, υπολογίζονται αναδρομικά τα σύνολα $T(t(s, f, 0), d - 1)$ και $T(t(s, f, 1), d - 1)$, συνδυάζονται με άθροιση κατά συνιστώσα και διατηρούνται, μόνο οι μη κυριαρχούμενες λύσεις.

Έτσι διασφαλίζεται ότι η Pareto βελτιστοποίηση διατηρείται τοπικά και καθολικά σε όλο το δέντρο αποφάσεων.

0.3.4 Τεχνική Προσέγγιση

Συναρτήσεις κόστους και απώλειας

Η *συνάρτηση απώλειας* ορίζεται ως δείκτης επιτυχούς αναστροφής της πρόβλεψης, δηλαδή ελέγχει αν μια τροποποίηση οδηγεί το δείγμα από την ανεπιθύμητη στην επιθυμητή κλάση:

$$\ell_{\text{flip}}(x, a; y^\star) = \mathbf{1}\{f(x + a) \neq y^\star\}.$$

Η *συνάρτηση κόστους* βασίζεται στο *Maximum Percentile Shift (MPS)*, που μετρά τη μέγιστη μετατόπιση ποσοστού (percentile) ενός χαρακτηριστικού μετά την τροποποίηση:

$$c_{\text{MPS}}(x, a) = \max_{j \in A} \big(Q_j(x_j + a_j) - Q_j(x_j)\big),$$

όπου Q_j είναι η εμπειρική αθροιστική συνάρτηση κατανομής του χαρακτηριστικού j. Το MPS είναι ανεξάρτητο από την κλίμακα των τιμών και προσφέρει μια ρεαλιστική εκτίμηση της «προσπάθειας» αλλαγής, καθιστώντας το κατάλληλο για αξιολόγηση σε επίπεδο αξιολόγησης πληθυσμών.

Παράμετροι και περιορισμοί

Η μέθοδος επιτρέπει ρυθμίσεις που καθορίζουν τη μορφή του δέντρου, ο οποίες είναι το μέγιστο βάθος d, το ελάχιστο πλήθος δειγμάτων ανά φύλλο m, και το πλήθος μέγιστων επιτρεπτών αλλαγών ανά τροποποίηση. Ο σωστός συνδυασμός αυτών των παραμέτρων εξισορροπεί την ερμηνευσιμότητα και τη γενίκευση των λύσεων. Για πολύ βαθιά δέντρα, το υπολογιστικό φορτίο κατά την επίλυση αυξάνεται εκθετικά και γι' αυτό χρησιμοποιείται μηχανισμός *timeout* που σταματά την αναζήτηση και επιστρέφει τη βέλτιστη λύση μέχρι τη στιγμή διακοπής. Αυτό βέβαια είναι εξίσου παραμετροποιήσιμο και στις λύσεις που παρουσιάζει η εργασία δεν ενεργοποιήθηκε αυτή η παράμετρος.

Υλοποίηση του συνόλου Pareto

Το σύνολο Pareto διαμορφώνεται μέσω της μερικής διάταξης των λύσεων στο επίπεδο (C, L), με εφαρμογή παραμέτρων *χαλάρωσης* ε για αριθμητική σταθερότητα:

$$x \prec_\varepsilon y \iff \forall i \in \{1, 2\}\ f_i(x) \leq f_i(y) + \varepsilon,\ \ \exists j : f_j(x) < f_j(y) - \varepsilon.$$

Η μέθοδος χρησιμοποιεί άνω και κάτω φράγματα που περιορίζουν το χώρο αναζήτησης, απορρίπτοντας πρόωρα μη αποδοτικές λύσεις. Ο αριθμός των διατηρούμενων μη κυριαρχούμενων δέντρων ελέγχεται από παράμετρο μεγέθους του συνόλου Pareto, διατηρώντας τη διαδικασία υπολογιστικά εφικτή. Μετά τον υπολογισμό των βέλτιστων φύλλων, οι συναρτήσεις *merge & add* του πλαισίου συνδυάζουν τα υποδέντρα, κρατώντας μόνο τις λύσεις που ανήκουν στο μη-κυριαρχούμενο σύνολο.

Συνολική παρατήρηση

Η παραπάνω διαδικασία καθιστά τη μέθοδο **SOGAR** υπολογιστικά σταθερή, διαθέσιμη προς αναπαραγωγή και ικανή να αποδίδει καθολικά βέλτιστες λύσεις. Με τη συνδυαστική χρήση της δυαδικής τροποποίησης, των προϋπολογισμένων τιμών και της Pareto βελτιστοποίησης, το σύστημα επιτυγχάνει αποδοτική ισορροπία μεταξύ πολυπλοκότητας και ακρίβειας, προσφέροντας πρακτική εφαρμοσιμότητα σε μεγάλα πινακοειδή σύνολα.

0.3.5 Προτεινόμενη Διαδικασία

Η μέθοδος **SOGAR** (Summaries of Optimal and Global Actionable Recourse) αποτελεί μια διατύπωση του προβλήματος των *Actionable Recourse Summaries* με αξιοποίηση του πλαισίου δυναμικού προγραμματισμού *STreeD* για βέλτιστα δέντρα αποφάσεων. Αφού έχει οριστεί το θεωρητικό πλαίσιο του προβλήματος, η ενότητα αυτή περιγράφει την πρακτική εκτέλεση της διαδικασίας, όπως αυτή απεικονίζεται στο Σχήμα 0.3.1. Η διαδικασία διαρθρώνεται σε τρία πλήρως αλληλεξαρτώμενα μέρη, από τα αρχικά δεδομένα εισόδου έως την παραγωγή του τελικού συνόλου Pareto.

1. Εκπαίδευση και πρόβλεψη

Αρχικά, εκπαιδεύεται ένας σταθερός *μη ερμηνεύσιμος ταξινομητής* f στο σύνολο εκπαίδευσης D_{tr} και πραγματοποιούνται προβλέψεις στα σύνολα $D_{\text{tr}} \cup D_{\text{te}}$. Το αποτέλεσμα είναι ένα επαυξημένο σύνολο με τις προβλεπόμενες ετικέτες $\hat{y} = f(x)$. Από αυτό εξάγεται ο πληθυσμός των αρνητικά ταξινομημένων δειγμάτων $X = \{x : f(x) = 0\}$, δηλαδή εκείνων που έλαβαν την ανεπιθύμητη πρόβλεψη και αποτελούν το αντικείμενο της διαδικασίας του actionable recourse.

2. Δυαδική τροποποίηση και δημιουργία μετα-δεδομένων

Στη συνέχεια, τα δεδομένα μετατρέπονται σε πλήρως δυαδικά τροποποιημένη μορφή D_{bin}, μέσω κωδικοποίησης *one-hot* για τα χαρακτηριστικά που παραπέμπουν σε κατηγορίες ή κατωφλίων για αριθμητικά χαρακτηριστικά με μεγάλο εύρος τιμών, διατηρώντας την αντιστοίχιση των δειγμάτων του X. Κατά τη διαδικασία αυτή δημιουργείται ένα *context αρχείο* C που αποθηκεύει την "απεικόνιση" μεταξύ αρχικών και δυαδικών χαρακτηριστικών, τα όρια τιμών και τους περιορισμούς μεταβλητότητας. Το αρχείο αυτό επιτρέπει τη σύνδεση των αρχικών δεδομένων με το δυαδικά τροποποιημένος σύνολο και αποτελεί απαραίτητη είσοδο για τα επόμενα βήματα.

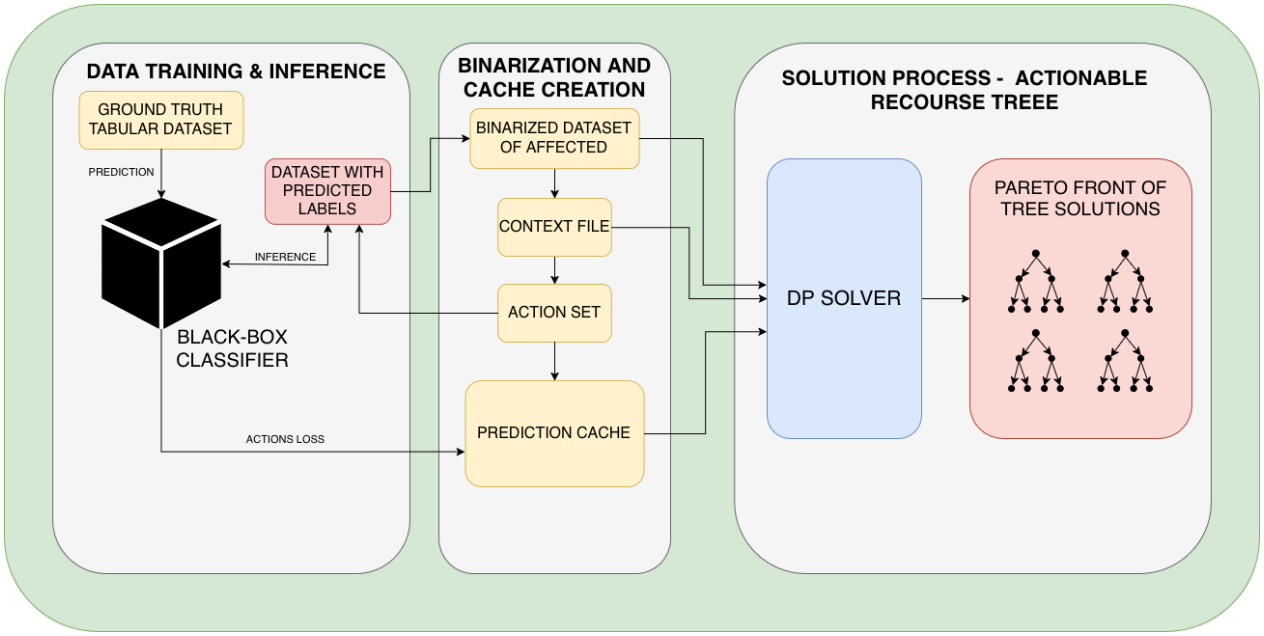

Figure 0.3.1: Αναπαράσταση διαδικασία μεθόδου SOGAR

Στην σχηματική αναπαράσταση της διαδικασίας του SOGAR, παρατηρούμε τον διαχωρισμό σε τρεις επί μέρους τομείς, οι οποίοι αλληλεπιδρούν μεταξύ τους κατά την διάρκεια της διαδικασίας.

3. Δημιουργία συνόλου ενεργειών

Με βάση το C, κατασκευάζεται ένα καθολικά εφικτό σύνολο ενεργειών $\mathcal{A}(x)$ για όλα τα δείγματα εισόδου, το οποίο συμμορφώνεται με τους περιορισμούς μεταβλητότητας και τον μέγιστο αριθμό επιτρεπτών αλλαγών ($k \leq 3$). Κάθε ενέργεια καθορίζεται στους αρχικούς άξονες χαρακτηριστικών και έχει ως σκοπό την αξιολόγηση έναντι του ταξινομητή f.

4. A priori υπολογισμός κόστους και προβλέψεων

Για κάθε $x \in A$ και $a \in \mathcal{A}(x)$, υπολογίζονται εκ των προτέρων:

$$\text{cost}(x, a) = c(a|x), \quad \text{flip}(x, a) = \mathbf{1}\{f(x + a) = 1\}.$$

Οι τιμές αυτές αποθηκεύονται στην *cache* με κλειδιά (x, a), ώστε να αποφεύγεται η επιβάρυνση κατά την επίλυση του προβλήματος, με επαναλαμβανόμενους υπολογισμούς των ίδιων τιμών κόστους και την συνεχόμενη ανταλλαγή δεδομένων μεταξύ ταξινομητή και του μοντέλου μας. Με τον τρόπο αυτό, ο αλγόριθμος εκτελεί αναζητήσεις κόστους και αποτελεσμάτων μέσω απλής ανάκτησης από τη μνήμη σε χρονική πολυπλοκότητα $O(1)$.

5. Επίλυση δέντρου βέλτιστων ενεργειών

Το επόμενο βήμα περιλαμβάνει την εισαγωγή των D_{bin}, C και του αρχείου της *cache* στον solver του **STreeD**. Ο αλγόριθμος εκτελεί την αναδρομική διαδικασία αναζήτησης και επίλυσης υπό περιορισμούς μέγιστου βάθους d, ελάχιστου μεγέθους φύλλου m και μέγιστου μεγέθους του συνόλου Pareto. Κάθε φύλλο λαμβάνει μία ενέργεια, και η αναζήτηση των βέλτιστων λύσεων γίνεται με χρήση παραμέτρων χαλάρωσης ε για συγκρίσεις κυριαρχίας και την οριοθέτηση άνω και κάτω φραγμάτων που περιορίζουν τον χώρο αναζήτησης από χειρότερες λύσεις. Η μέθοδος παράγει μετά από εξαντλητική αναζήτηση, όλα τα δυνατά υποδέντρα μέχρι το δεδομένο βάθος, επιστρέφοντας όλες τις μη κυριαρχούμενες λύσεις.

6. Παραγωγή του συνόλου Pareto

Η έξοδος του solver είναι το σύνολο Pareto $T^\star$ των μη κυριαρχούμενων δέντρων. Κάθε δέντρο διαχωρίζει τον χώρο X χωρίς επικαλύψεις ίδιων δειγμάτων σε πολλαπλές περιοχές του χώρου και αντιστοιχίζει μία εφικτή τροποποίηση ανά φύλλο, μαζί με τα συνολικά στατιστικά κόστους και επιτυχών αναστροφών. Αυτά αποτελούν τις τελικές actionable recourse summaries του **SOGAR**.

Πλεονεκτήματα και προσαρμογές

Η συγκεκριμένη προσέγγιση δυναμικού προγραμματισμού, παρέχει καθολική βελτιστότητα με πλήρως ντετερμινιστικό τρόπο, καθώς εξετάζονται όλες οι πιθανές περιπτώσεις φύλλων. Η μέθοδος εκμεταλλεύεται πλήρως τους μηχανισμούς της *cache* και τα άνω και κάτω φράγματα του **STreeD**, επιτυγχάνοντας σημαντική μείωση του χρόνου εκτέλεσης χωρίς αλλοίωση της βέλτιστης απόδοσης.

Επιπλέον, η μέθοδος δεν επιστρέφει μόνο μία λύση, αλλά ένα σύνολο Pareto ισοδύναμα βέλτιστων λύσεων. Ο χρήστης μπορεί να επιλέξει ανάλογα με τις ανάγκες του πεδίου ορισμού των δεδομένων τους, είτε δέντρα που δίνουν έμφαση στο χαμηλό κόστος, είτε δέντρα που στοχεύουν σε υψηλότερα ποσοστά επιτυχών αναστροφών στην επιθυμητή ετικέτα. Με αυτόν τον τρόπο, η πολλαπλότητα των λύσεων δεν περιπλέκει τη λήψη αποφάσεων αλλά την ενισχύει, προσφέροντας διαφάνεια, ευελιξία και δυνατότητα ερμηνείας των προτεινόμενων τροποποιήσεων για διαφορετικές πληθυσμιακές ομάδες.

0.4 Πειραματικό Μέρος

0.4.1 Πειραματική Προετοιμασία

- Η υλοποίηση εκτελείται σε Apple Silicon M4 με 16 GB RAM.
- Το **SOGAR** υλοποιείται *STreeD* και ενσωματώνεται στον solver του, με το μέρος του solver υλοποιημένο σε C++17.
- Η προεπεξεργασία και το caching υλοποιούνται σε Python 3.12
- Χρησιμοποιήσαμε τον ταξινομητή LightGBM[30], με τις παραμέτρους που χρησιμοποιούν οι σχετικές έρευνες που συγκρινόμαστε

0.4.2 Σύνολα Δεδομένων και Προεπεξεργασία

Χρησιμοποιούμε τέσσερα σύνολα δεδομένων που χρησιμοποιούνται σε ελέγχους αποφάσεων υψηλού ρίσκου, κατάλληλα για την διαδικασία του Actionable Recourse, τα οποία είναι το German Credit[31], το Bank Marketing [32], το Adult Income [33] και το IBH HR Employee Attrition [34]. Το Adult υποδειγματοληπτείται στο μισό τους αρχικού του μεγέθους για να ολοκληρώνονται όλα τα baselines με διατήρηση των αρχικών λόγων μεταξύ των κλάσεων, ενώ για το Bank χρησιμοποιούμε τη δημόσια εκδοχή υποσυνόλου η οποία παραπέμπει στο 10% του πρωτοτύπου.

Προεπεξεργασία

Τα χαρακτηριστικά που παραπέμπουν σε κατηγορίες κωδικοποιούνται σε one-hot κωδικοποίηση όπως στο CET. Τα χαρακτηριστικά με συνεχείς τιμές τροποποιούνται δυαδικά σε διαστήματα βάσει ποσοστιαίων ορίων συμβατών με το *STreeD*. Όπου είναι εφικτό, ακέραιες μεταβλητές τροποποιούνται δυαδικά με ένα bin ανά τιμή. Περιορίζουμε τη διόγκωση της διάστασης του χώρου χαρακτηριστικών, μετά τη διαδικασία δυαδικής τροποποίησης, ώστε να μην προκληθεί αλλοίωση του χώρου των χαρακτηριστικών. Εμπειρικά αυτό δεν χειροτερεύει την ποιότητα λύσεων και είναι αναγκαίο για ορθούς δυαδικούς διαχωρισμούς στον solver.

Μεταβλητότητα και σύνολο τροποποιήσεων

Χαρακτηριστικά όπως ηλικία, φύλο, φυλή, οικογενειακή κατάσταση, σχέση και χώρα σημειώνονται ως αμετάβλητα ανά dataset για την αποφυγή διακρίσεων και τον σεβασμό των προκαθορισμένων περιορισμών μεταβλητότητας. Το σύνολο τροποποιήσεων ανά δείγμα περιορίζεται σε σπανιότητα αλλαγών $k \leq 3$, σε συμφωνία με τη σχετική βιβλιογραφία και την κατάλληλη σύγκριση.

Πολιτική binning και ευσταθές κόστος

Για μεγάλες αριθμητικές κλίμακες, όπως το εισόδημα, χρησιμοποιούμε 20 έως 50 διαστήματα περίπου ίσου μήκους και μεταφράζουμε έναν επιλεγμένο διαχωρισμό ως κίνηση προς τη διάμεσο του bin στόχου που προτείνει μια τροποποίηση. Έτσι προκύπτουν αλλαγές με ρεαλιστικά μεγέθη, όπως +500, +1500, +3000, αντί για υπερβολικά άλματα ή υπερεκτιμημένες κινήσεις μηδαμινής ουσιαστικής αλλαγής. Τα δυαδικά χαρακτηριστικά και οι κατηγορίες αντιστοιχούν σε ενεργοποίηση ή αλλαγή κατηγορίας, σύμφωνα με τους κανόνες μεταβλητότητας. Το σύνολο των τροποποιήσεων (cache) είναι μεγαλύτερο από αυτό προηγούμενων εργασιών, αλλά επειδή όλα τα κόστη και οι flip εκβάσεις προϋπολογίζονται, οι αξιολογήσεις φύλλων κατά την αναζήτηση *DP* γίνονται με άμεσες ανακτήσεις τιμών, read-only lookups στην cache.

0.4.3 Μετρικές Αξιολόγησης

Flip loss (Αναστροφή ετικέτας)

Για στόχο $y^\star = 1$ και μια υποψήφια τροποποίηση a σε δείγμα εισόδου x, ο δείκτης αποτυχίας αναστροφής είναι

$$\ell_{\text{flip}}(x, a; y^\star) = \mathbf{1}\{f(x + a) \neq y^\star\}. \tag{0.4.1}$$

Σε επίπεδο δέντρου τ και πληθυσμού $D_0 = \{x : f(x) = 0\}$,

$$L(\tau) = \sum_{x \in D_0} \ell_{\text{flip}}\big(x, a(\ell(x)); y^\star\big). \tag{0.4.2}$$

Κόστος MPS (Maximum Percentile Shift)

Με Q_j την εμπειρική CDF του χαρακτηριστικού j στον αρχικό χώρο τιμών και $x' = x + a$,

$$c_{\text{MPS}}(x, a) = \max_{j \in A(x)} \Big(Q_j(x'_j) - Q_j(x_j)\Big), \tag{0.4.3}$$

και σε επίπεδο δέντρου

$$C(\tau) = \sum_{x \in D_0} c_{\text{MPS}}\big(x,\, a(\ell(x))\big). \tag{0.4.4}$$

Χρήση στα πειράματα

- **Συμβατότητα:** Χρησιμοποιούμε τις μετρικές τις (0.4.1)–(0.4.3) όπως στο CET για άμεση συγκρισιμότητα. Τα Q_j υπολογίζονται μια φορά σε train+test μετά τις προβλέψεις του ταξινομητή και αποθηκεύονται στο *context*.
- **Υπολογισμός κόστους και συνάρτησης απώλειας:** Αν και το *STreeD* λειτουργεί σε δυαδοποιημένα γνωρίσματα, τα κόστη (0.4.3) και τα $L(\tau), C(\tau)$ (0.4.2)–(0.4.4) υπολογίζονται στον *αρχικό* χώρο τιμών όπου και εκπαιδεύεται ο ταξινομητής.
- **Ταίριασμα από bins σε αρχικά χαρακτηριστικά:** Για μεγάλου εύρους γνωρίσματα, η επιλογή bin μεταφράζεται σε τιμή $\text{med}_{b^\star}$ (διάμεσος του στόχου bin) πριν μπει στην Q_j ώστε να αποφεύγονται αστάθειες και να παρέχεται όση τον δυνατόν περισσότερη ευρωστία.
- **Αθροιστικότητα:** Και οι δύο συναρτήσεις στόχοι είναι *προσθετικοί ανά φύλλο* και αθροίζονται μόνο πάνω στο D_0 (δείγματα εξεταζόμενου πληθυσμού).
- **Αριθμητική ανοχή:** Οι συγκρίσεις κυριαρχίας γίνονται με παραμέτρους χαλάρωσης ε για σταθερότητα σε συγκρίσεις μεταξύ αριθμών κινητής υποδιαστολής.
- **Αποδοτικότητα:** Τα ζεύγη (x, a) έχουν προϋπολογισμένα $\ell_{\text{flip}}, c_{\text{MPS}}$, άρα οι αξιολογήσεις φύλλων γίνονται με $O(1)$ αναγνώσεις από cache.

0.4.4 Αξιολόγηση Λύσεων

Κοινή ρύθμιση αξιολόγησης

Όλες οι μέθοδοι αξιολογούνται με τους ίδιους ορισμούς κόστους (MPS) και *flip loss*. Θέτουμε $\gamma = 0.99$ (σχεδόν μηδαμινή μείωση της βαρύτητας του loss). Στο **SOGAR** δεν χρησιμοποιείται ο κανονικοποιητής λ του CET, επειδή ελέγχουμε άμεσα τη δομή με *μέγιστο βάθος* d, *ελάχιστο μέγεθος φύλλου* m, *μέγιστο πλήθος εσωτερικών κόμβων* M. Για το CET και Cluster-wise actionable recourse χρησιμοποιήθηκε ο solver **Gurobi**[1] αντί του αρχικά ορισμένο **CPLEX**[2] για μεγαλύτερη συμβατότητα υλοποίησης, βάσει των υπολογιστικών μας πόρων. Για την μέθοδο του **AReS** δεν ήταν εφικτό να τρέξει στο *Adult Income* (ακόμη και με υποδειγματοληψία) λόγω έλλειψης υπολογιστικής ισχύος για ένα πολύ επιβαρυντικό και υπολογιστικά απαιτητικό πρόβλημα.

Επέκταση και pruning του χώρου λύσεων

Το πλήθος των μη-κυριαρχούμενων λύσεων εξαρτάται έντονα από τις παραμέτρους (d, M, m), που αναπαριστούν το βάθος, τους εσωτερικούς κόμβους και τον ελάχιστο αριθμό δειγμάτων ανά φύλλο αντίστοιχα, τα όρια του συνόλου Pareto που περιορίζονται από το ε και το μέγεθος του action set.

- *Ρηχά δέντρα* (π.χ. $d = 2, M = 3$) σε μικρά/μεσαία σύνολα: $\sim 180-220$ λύσεις. Αύξηση d, M οδηγεί σε $\sim 250-500$.
- *Μεσαίο μέγεθος* (π.χ. Bank): παρόμοια κλίμακα λύσεων για ρηχά δέντρα, αυξάνει με το action set.
- *Μεγάλα σύνολα* (Adult): $\sim 2000-3000$ λύσεις, ακόμη και για $d = 2$ ή 3.

Τεχνικές ελέγχου pruning:

- **Κανόνες συγχώνευσης** στο *STreeD* που περιορίζουν τους συνδυασμούς υποδέντρων.
- παράμετρος **χαλάρωσης** ε στην κυριαρχία για λιγότερες και σταθερότερες συγκρίσεις (αποτελεσματικό, διατηρεί ορθότητα).

Φύση και συνέπεια λύσεων

Οι δομές δέντρων παραμένουν ερμηνεύσιμες και οι τροποποιήσεις ανά φύλλο διατηρούν την διαφάνεια, μέσω πλήρους κάλυψης του εξεταζόμενου πληθυσμού D_0).

- **Γενίκευση:**

[1] https://www.gurobi.com/solutions/gurobi-optimizer/
[2] https://www.ibm.com/products/ilog-cplex-optimization-studio

  - Ρηχότερα δέντρα δίνουν παρόμοια αποτελέσματα σε train/test κόστος και loss.
  - Βαθύτερα δέντρα βελτιώνουν το train αλλά χειροτερεύουν το test (κλασικό bias–variance).

- **Σταθερότητα Pareto κορυφής:** Οι κορυφαίες 10 λύσεις διαφέρουν *ελάχιστα* σε invalidity, σε τάξη μεγέθους 10^{-3} ανά βήμα, προσφέροντας *πολλαπλές σχεδόν ισοδύναμες* επιλογές με διαφορετική δομή και τροποποιήσεις.

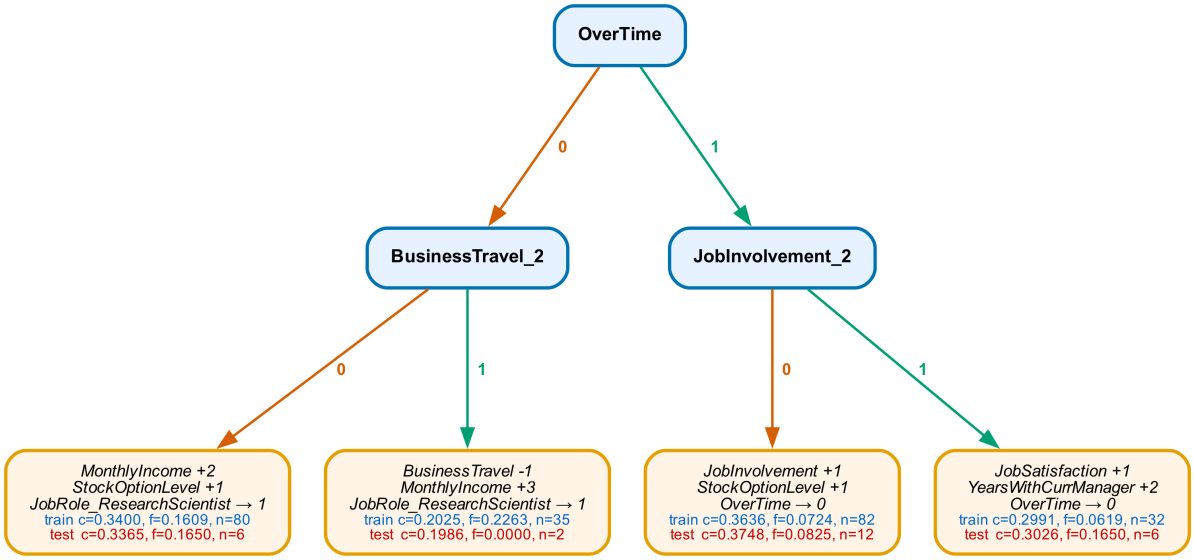

Figure 0.4.1: Αναπαράσταση Δεντρικής Σύνοψης, βάθους $d = 2$

Οπτικοποίηση Pareto και ερμηνεία

Το μέτωπο εμφανίζει την αναμενόμενη *αντιστρόφως ανάλογη* σχέση κόστους και loss, αφού χαμηλό κόστος ⇒ υψηλότερες αποτυχίες flip και αντιστρόφως. Η θέση του «βέλτιστου» εξαρτάται από την επιλογή του γ.

- **Επιλογή λύσης:** Πέρα από το invalidity, μπορούν να χρησιμοποιηθούν άλλες μετρικές όπως η απόσταση από την αρχή στο επίπεδο (C, L) ως εναλλακτικό κριτήριο, ανάλογα με το πλαίσιο.

- **Γενίκευση στο test:** Τα test σημεία προσεγγίζουν το σχήμα της θεωρητικά απεικονιζόμενης καμπύλης του συνόλου Pareto που προκύπτει μετά το training της μεθόδου (μικρές αποκλίσεις λόγω μικρού test set), υποδηλώνοντας *συνεπή trade-off* και στα δύο σύνολα.

0.4.5 Σχολιασμός Πειραματικών Αποτελεσμάτων

Τα πειράματα που τρέξαμε αφορούσαν ένα 10-fold cross validation στα τέσσερα εξεταζόμενα σύνολα δεδομένων. Χρησιμοποιήσαμε τις προεπιλεγμένες παραμέτρους για όλες τις συγκρινόμενες μεθόδους όπως είχαν προκαθοριστεί από την δημοσιευμένη υλοποίηση του CET. Για το CET προχωρήσαμε στην εξής αλλαγή που αφορά μόνο τα πειράματα στο Adult dataset, μειώνοντας τον αριθμό των επαναλήψεων στις $T = 1000$ αφού η εκτέλεση του αλγορίθμου για το συγκεκριμένο απαιτητικό dataset ήταν οριακά απαγορευτική όσον αφορά τον χρόνο εκτέλεσης. Για το AReS προσπαθήσαμε με διαφορετική παραμετροποίηση να τρέξουμε πειράματα στο ίδιο προαναφερθέν σύνολο δεδομένων, αλλά δεν αντλήσαμε κανένα αποτέλεσμα.

Για την δική μας μέθοδο το SOGAR, τα πειράματα που τρέξαμε ήταν για δύο ζεύγη παραμέτρων, $d = 2, n = 3$ και $d = 3, n = 7$, για το μέγιστο βάθος και τον μέγιστο αριθμό εσωτερικών κόμβων. Η κοινά ορισμένη υπερπαράμετρος ήταν το $\gamma = 0.99$ σε κάθε μέθοδο.

Κύρια ευρήματα

- Το SOGAR βελτιώνει σημαντικά το invalidity score έναντι των CET και AReS σε όλα τα σύνολα.
- η μέση μείωση είναι τάξης $\approx 40\%$ (train & test), προερχόμενη κυρίως από δραστικά χαμηλότερη συνάρτηση απώλειας (flip loss) σε παρόμοιο κόστος.
- Attrition, German: μεγάλη πτώση στο loss με παρεμφερές κόστος $\rightarrow$ ισχυρή μείωση invalidity.
- Bank: μικρές αλλά συνεπείς βελτιώσεις και σε κόστος και σε loss.
- Adult: καλύτερη επίδοση του SOGAR, αλλά δύσκολη σύγκριση η οποία αποδίδεται με το χαμηλότερο T του CET.

Σύντομη επεξήγηση

- **Υπολογισμός όλων τον πιθανών φύλλων:** Η μέθοδος του δυναμικού προγραμματισμού με αναζήτηση λύσεων συνόλων Pareto μέσω του STreeD εξετάζει τα φύλλα ώστε να ελαχιστοποιεί τη συνάρτηση απώλειας για δεδομένο κόστος, για κάθε πιθανή τροποποίηση που περιέχεται στο σύνολο τροποποιήσεων.
- **Σύγκριση κυριαρχίας λύσεων με την παράμετρο ϵ και BnB:** Αποκόπτει νωρίς αριθμητικά ισοδύναμες λύσεις, διατηρώντας μόνο ουσιωδώς καλύτερες και σταθερές δομές.
- **Κόστος και ποιότητα:** Ελαφρώς υψηλότερο μέσο κόστος σε ορισμένα folds οφείλεται στην απεικόνιση συνεχών χαρακτηριστικών με την χρήση της διαμέσου που αναφέραμε προηγουμένως, και δεν υποβαθμίζει την ποιότητα της λύσης.
- **Γενίκευση:** οι κορυφαίες λύσεις είναι οριακά ισόπαλες σε invalidity και προσφέρουν εναλλακτικές με παρόμοια επίδοση αλλά διαφορετική ερμηνευσιμότητα και πολιτική κόστους.

Χρόνος, ευαισθησίες και συμπέρασμα

- **Χρόνος σε μικρά/μεσαία σύνολα δεδομένων:** Συγκρίσιμος με CET και AReS, συχνά ελαφρώς μεγαλύτερος λόγω μεγαλύτερης συνδυαστικής ένωσης των επιμέρους φύλλων.
- **Bank:** Αναμενόμενη μείωση της επίδοσης λόγω μεγαλύτερου χώρου αναζήτησης.
- **Adult:** Ταχύτερο από CET στην πράξη, επειδή το CET κλιμακώνει άσχημα με T και την χρήση του MILO ανά φύλλο, ενώ το STreeD με caches έχει ανακτήσεις $O(1)$ ανά (x, a).
- **Υλοποίηση:** Όλα είναι υλοποιημένα single-threaded, συνεπώς, παραλληλοποίηση cache και συγκρίσεων αναμένεται να μειώσει ουσιαστικά τον χρόνο.
- **Δυαδική τροποποίηση δεδομένων εισόδου:** Το binning επηρεάζει την γεωμετρία των τροποποιήσεων και την μετρική MPS.
- **Συμπέρασμα:** Το SOGAR αξιοποιεί πλήρως την καθολική βελτιστοποίηση δέντρου και την πλήρη αξιολόγηση εφικτών τροποποιήσεων με στόχο το χαμηλότερο invalidity και την υψηλή ερμηνευσιμότητα.

0.5 Συμπεράσματα

Το *Actionable Recourse* αποτελεί έναν πρακτικό και ουσιαστικό τρόπο αντιμετώπισης της αδιαφάνειας των μη ερμηνεύσιμων μοντέλων που επιτυγχάνουν κορυφαίες επιδόσεις στην διαδικασία πρόβλεψης. Επιτρέπει την κατανόηση όχι μόνο του αποτελέσματος του μοντέλου, αλλά και τις μεταβολές των χαρακτηριστικών που πρέπει να συμβούν για την επίτευξη της επιθυμητής πρόβλεψης.

Σε αυτήν την διπλωματική εργασία αναλύουμε διεξοδικά το πρόβλημα του *Actionable Recourse* και αξιοποιούμε την ιδέα των δεντρικών συνοψίσεων που εισάγει το *CET*, μέσω των βέλτιστων δεντρικών δομών. Η ανασκόπηση των αλγορίθμων δέντρων αποφάσεων αναδεικνύει τα πλεονεκτήματα ερμηνευσιμότητας, και ταυτόχρονα τη διαφορά των βέλτιστων δέντρων αποφάσεων, όπου η *καθολική βελτιστότητα* καθιστά τους αλγορίθμους αυτού του είδους κατάλληλους για αυστηρή βελτιστοποίηση με τα κριτήρια της *διαχωρισιμότητας*. Αυτό μας οδήγησε

Table 1: Πειραματικά αποτελέσματα σε 10-fold cross-validation

		Train			Test		
Dataset	**Method**	**Cost**	**Loss**	**Inv.**	**Cost**	**Loss**	**Inv.**
Attrition	Clustering	0.068 ± 0.05	0.940 ± 0.04	0.999 ± 0.01	0.0084 ± 0.11	0.918 ± 0.13	0.992 ± 0.03
	AReS	0.442 ± 0.05	0.439 ± 0.09	0.877 ± 0.06	0.445 ± 0.07	0.285 ± 0.09	0.728 ± 0.08
	CET	0.294 ± 0.12	0.477 ± 0.15	0.776 ± 0.07	0.305 ± 0.12	0.45± 0.21	0.751 ± 0.14
	SOGAR	0.302 ± 0.032	0.135 ± 0.03	↓ **0.437 ± 0.035**	0.295 ± 0.021	0.1297 ± 0.060	↓ **0.425 ± 0.066**
German	Clustering	0.042 ± 0.02	0.916 ± 0.04	0.949 ± 0.03	0.046 ± 0.02	0.925 ± 0.05	0.962 ± 0.03
	AReS	0.452 ± 0.09	0.232 ± 0.05	0.683 ± 0.11	0.467 ± 0.12	0.265 ± 0.08	0.732 ± 0.14
	CET	0.107 ± 0.02	0.269 ± 0.01	0.374 ± 0.07	0.108 ± 0.02	0.226 ± 0.11	0.332 ± 0.11
	SOGAR	0.109 ± 0.007	0.086 ± 0.022	↓ **0.195 ± 0.019**	0.109 ± 0.013	0.111 ± 0.058	↓ **0.220 ± 0.059**
Bank	Clustering	0.010 ± 0.02	0.993 ± 0.01	0.993 ± 0.012	0.607 ± 0.010	0.994 ± 0.011	0.994 ± 0.0
	AReS	0.361 ± 0.02	0.799 ± 0.08	1.152 ± 0.09	0.363 ± 0.03	0.798 ± 0.08	1.152 ± 0.08
	CET	0.035 ± 0.003	0.018 ± 0.01	0.053 ± 0.01	0.035 ± 0.01	0.019 ± 0.01	0.054 ± 0.01
	SOGAR	0.029 ± 0.0004	0.007 ± 0.006	↓ **0.037 ± 0.006**	0.029 ± 0.0006	0.011 ± 0.007	↓ **0.039 ± 0.007**
Adult	Clustering	0.95 ± 0.0	0.107±0.01	1.056± 0.01	0.948± 0.04	0.094±0.06	1.041±0.08
	AReS	—	—	—	—	—	—
	CET	0.935 ± 0.04	0.110±0.012	1.045± 0.02	0.94± 0.04	0.099±0.05	1.039±0.08
	SOGAR	0.166 ± 0.005	0.448 ± 0.007	↓ **0.626 ± 0.002**	0.172 ± 0.005	0.435 ± 0.002	↓ **0.607 ± 0.002**

Η πειραματική διαδικαστία του 10-fold cross-validation με την χρήστη του ταξινομητή LightGBM [30].

στην προσέγγιση δυναμικού προγραμματισμού και στην μέθοδο του *STreeD*, στο οποίο προσαρμόσαμε τη δι-*αντικειμενική* διατύπωση του προβλήματος μας.

Η προτεινόμενη μέθοδος μας, το **SOGAR**, θέτει μια στέρεη θεωρητική βάση για τη συστηματική κατασκευή συνόλων *Pareto* από μη κυριαρχούμενες, καθολικά βέλτιστες λύσεις δεντρικών δομών. Πειραματικά, σε πρότυπα πινακοειδή σύνολα, καταδεικνύεται αύξηση της αποτελεσματικότητας των ενεργειών μέσω νέων διαχωρισμών των δειγμάτων εισόδου σε φύλλα που προκύπτουν με βέλτιστο τρόπο μέσω την μεθόδου του δυναμικού προγραμματισμού. Επιπλέον, η αξιολόγηση ολόκληρου του συνόλου εφικτών τροποποιήσεων ανά υποψήφιο φύλλο, παρότι είναι υπολογιστικά απαιτητική, παραμένει συγκρίσιμη με τη σχετική βιβλιογραφία και αναδεικνύει καθολικά βέλτιστες λύσεις.

Η δι-αντικειμενική διατύπωση επιτρέπει την ανάκτηση πολλαπλών λύσεων ανά εκτέλεση της μεθόδου, παρέχοντας ευελιξία επιλογής μεταξύ λύσεων με σχεδόν ισοδύναμη επίδοση. Συνολικά, τα *Actionable Recourse Summaries* με την υλοποίηση βέλτιστης δεντρικής δομής, επιστρέφουν ένα διαφανές και καθολικά βέλτιστο σύνολο πολιτικών recourse λύσεων για κάθε δείγμα του εξεταζόμενου πληθυσμού. Πετυχαίνοντας, έτσι, βελτίωση των επιδόσεων συγκριτικά με της σχετικές έρευνες, με την εγγύηση της διατήρησης της ερμηνευσιμότητας.

Μελλοντική Έρευνα

- **Παράλληλος υπολογισμός:** multi-threaded δημιουργία cache και παράλληλη αξιολόγηση φύλλων εντός του STreeD για την μείωση υπολογιστικού χρόνου.
- **Συνεχή χαρακτηριστικά:** διατύπωση που χειρίζεται συνεχείς μεταβλητές χαρακτηριστικών χωρίς πλήρη δυαδική τροποποίηση, διατηρώντας τη δι-αντικειμενική βελτιστοποίηση.
- **Ευρύτερη αξιολόγηση:** Πειράματα σε περισσότερα σύνολα και ταξινομητές υψηλής επίδοσης με διαφορετικά γνωρίσματα ως προς τα όρια και την σταθερότητα των αποφάσεων.
- **Δικαιοσύνη:** Ενσωμάτωση περιορισμών δικαιοσύνης ως πρόσθετων στόχων ώστε οι πολιτικές recourse να κατανέμονται ισότιμα σε ευαίσθητους και μη υποπληθυσμούς.

Chapter 1

Introduction

Artificial Intelligence (AI) has become integral to decision-making systems, influencing outcomes in domains such as credit allocation, employment opportunities, education, and healthcare. While predictive models have achieved remarkable accuracy, their complexity often renders them opaque to those most affected by their outputs. This complicated situation raises concerns about accountability and bias, particularly in high-stakes applications where algorithmic decisions can reshape the course of people's lives.

Much of the existing literature focuses on *instance-level counterfactuals*, generating explanations independently for everyone, understanding the locally minimal changes that would change the prediction of a model. Although informative, such explanations can lack global integrity and may produce contradictory recommendations for similar profiles, raising bias concerns that are difficult to generalize in a population of affected individuals. To overcome these limitations, recent research has proposed *recourse summaries*, which deliver structured and consistent recommendations across groups of instances. *Decision trees (DT)*, valued for their interpretability and widespread use, are well-suited to this summarization task.

A notable example, from which this work departs, is the framework of *Counterfactual Explanation Trees (CET)* [1], which assigns counterfactual actions to the leaves of a decision tree. CET marks an important step toward global recourse, yet it relies on heuristic local optimization per leaf and then averages the results to global objectives. This approach, even though it may seem to provide optimal solutions, it does not provide a globally optimal tree structure and therefore cannot guarantee global optimality of chosen actions and computed scores. This limitation raised the concern that an implementation of the task via Optimal Decision Tree structures, would successfully improve the existing solutions.

This thesis introduces a principled framework for *Actionable Recourse Summaries* using *optimal decision tree structures*. Our method uses the advantage of the dynamic programming framework of *Separable Trees with Dynamic Programming (STreeD)* [2], which provides globally optimal decision tree solutions under separability conditions. Specifically, the STreeD framework provides the opportunity to design any task that can be defined though the separable criteria, thus we create a new optimization task within the DP decision tree environment that assigns minimal-cost, high-success actions to tree leaves. The outcome of the task is as expected globally optimal and consistent actionable recourse summaries. The task is by its nature optimizing multiple objectives, balancing the *cost of actions* with the *rate of successful outcome flips*. For that reason, the framework has a valuable method to implement such tasks with ease, by formulating a Pareto-optimal output, by providing all the optimal non-dominated solutions, that provide a diversity of choices for the user to decide which partition of the solutions matches the needs of the task implemented. The type of solutions that this thesis finds more useful, either the goal is explainability or performance comparison with other works, is the minimal sum of the two individual objectives.

Contributions.

- We provide a systematic study of counterfactual explanations and actionable recourse and clarify their role within explainable AI and responsible machine learning.
- We introduce a novel optimization task within the STreeD framework for constructing *optimal recourse*

decision trees (DT), which assign consistent, minimal-cost actions to groups of instances with provable global optimality.

- We empirically validate the framework on benchmark tabular datasets, demonstrating improvements over CET baselines in both action cost and flip rate, while scaling to larger datasets and deeper tree structures.

Outline.

- Chapter 2 reviews the theoretical background of machine learning and supervised learning, with a focus on DT interpretability, and presents *Optimal Decision Trees (ODT)*, including existing methods and the dynamic programming approach of STreeD.
- Chapter 3 surveys the foundations of Explainability in AI, counterfactual explanations, Actionable Recourse and Actionable Recourse Summaries, and ends with the analysis of current state-of-the-art methods that influenced the development of this thesis.
- Chapter 4 introduces the problem setting, the limitations of related work and the proposed method for optimal recourse summaries, detailing the integration of the multi-objective optimization task via STreeD solver [2].
- Chapter 5 describes the experimental setup and presents results of benchmarking with compared reproducible methods.
- Finally, Chapter 6 concludes the thesis by summarizing contributions, identifying limitations, and outlining directions for future work.

Chapter 2

Machine Learning Background

The pursuit of constructing machines capable of reasoning, learning, and adaptation has influenced the development of engineering and computer science for several decades. The introduction of digital computation transformed this pursuit into the scientific field of Artificial Intelligence (AI), which focuses on designing systems that perform tasks requiring human-like intelligence, including perception, reasoning, and decision-making. Early AI systems primarily utilized symbolic methods, relying on explicitly defined rules and logical representations of knowledge. However, the complexity and uncertainty inherent in real-world problems demonstrated the limitations of hand-crafted rules. Consequently, data-driven approaches emerged, allowing machines to learn directly from empirical experience.

Machine Learning (ML) emerged as the subfield of AI devoted to learning from data rather than following explicit instructions. ML algorithms infer patterns and relationships from empirical evidence, enabling systems to improve their performance through experience [35, 36]. By learning statistical mappings between inputs and outputs, these models generalize from observed examples to unseen cases, allowing computers to build competence autonomously. Applications of ML range from medical diagnosis and financial forecasting to language translation and personalized recommendations. Depending on the nature of the input data, such as text, images, audio, or tabular attributes, learning systems are specialized into areas like *Computer Vision*, *Natural Language Processing*, *Speech Recognition*, and *Recommendation Systems*.

This chapter systematically examines the theoretical foundations of Machine Learning (ML). It begins by presenting the taxonomy of ML algorithms based on the degree of supervision during training, encompassing *supervised*, *unsupervised*, *semi-supervised*, *self-supervised*, and *reinforcement learning*. Within this framework, *Decision Trees (DT)* are situated in the supervised learning paradigm, as they derive predictive rules from labeled data to map input features to target outputs. Decision trees are notable for their interpretability and transparent logic, partitioning the feature space via simple, human-readable rules. This property renders decision trees among the most accessible and widely applied models in interpretable machine learning. Building on their foundational design, the subsequent sections detail the principles of decision tree construction, the historical evolution of training methodologies, and recent advances culminating in state-of-the-art models. Notably, *Optimal Decision Trees* [5] seek to generate globally optimal solutions under formal constraints, addressing challenges that have traditionally limited conventional tree-based approaches.

Contents

2.1 Learning Categories

Machine Learning algorithms are commonly categorized according to the type of supervision or experience provided during training. The main categories are *supervised*, *unsupervised*, and *reinforcement learning*, while *semi-supervised* and *self-supervised* learning can be viewed as intermediate or hybrid paradigms [37]. This taxonomy reflects the nature of the information the model receives to guide parameter updates and the extent to which feedback is explicit or implicit. In all cases, the goal of learning is to generalize beyond the observed data, capturing the underlying regularities that connect inputs to desired behaviors or representations. The balance between available supervision, model complexity, and data variability determines both the efficiency and reliability of learning, forming one of the central trade-offs in modern machine learning research.

Supervised Learning

In supervised learning, each example consists of an input vector $m\mathbf{x}$ and a corresponding outcome y, also called label or target. The objective is to learn a function $f(\mathbf{x})$ that accurately predicts y form $\mathbf{x}$ by minimizing a defined loss function over a set of labeled examples:

$$\min_{f} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i), \tag{2.1.1}$$

where $\ell(\cdot)$ measures the prediction error, such as squared loss for regression or cross-entropy loss for classification. This process can also be interpreted as estimating the conditional probability distribution $p(y|\mathbf{x})$. Common supervised tasks include classification, regression, and forecasting, which form the foundation of many predictive modeling applications. Supervised methods are especially relevant to interpretable models such as decision trees, where the learned function is represented through explicit, human-readable rules that relate input features to outcomes.

Unsupervised Learning

Unsupervised learning methods operate on data without associated labels. The goal is to discover underlying patterns or latent structures within the data by modeling its distribution $p(\mathbf{x})$. Typical tasks include clustering, which groups similar data points, and dimensionality reduction, which identifies compact low-dimensional representations that preserve essential relationships within the data. Such methods often serve as exploratory tools or as pre-processing stages that reveal intrinsic structure before applying downstream supervised algorithms. Frequently, an unsupervised algorithm aims to find parameters θ by maximizing likelihood or a related alternate metric:

$$\max_{\theta} \sum_{i=1}^{n} \log p(\mathbf{x}_i; \theta).$$

A prominent example is the pretraining of (auto-regressive) Large Language Models (LLMs), which are trained by *next-token maximum likelihood* on unlabeled collections of texts, such as GPT-3 and GPT-4 [38]

Semi-supervised Learning

Semi-supervised learning occupies a middle ground between supervised and unsupervised approaches. In this setting, the training data is composed of a small subset of labeled examples and a large pool of unlabeled ones. By leveraging the structure of unlabeled data alongside limited supervision, semi-supervised models can enhance generalization performance, particularly when acquiring labels is expensive or impractical. Many modern algorithms employ consistency regularization or pseudo-labeling strategies to leverage the information embedded in unlabeled samples, optimizing a combined objective:

$$\mathcal{L} = \mathcal{L}_{\text{sup}} + \lambda \, \mathcal{L}_{\text{unsup}},$$

where λ balances the contribution of the unsupervised consistency term.

Self-supervised Learning

Self-supervised learning is closely related to semi-supervised learning but does not rely on any human-labeled data. Instead, it derives supervision signals directly from the data itself through pretext tasks designed to learn meaningful internal representations. Once trained, these representations can be transferred to downstream tasks such as classification, detection, or segmentation. Self-supervision has recently gained prominence as a scalable paradigm for representation learning in domains like vision and natural language, where constructing explicit labels for massive datasets is infeasible.

Reinforcement Learning

Reinforcement learning differs fundamentally from the above categories, as it involves learning through interaction with an environment rather than from a static dataset. An agent observes a state, performs an action, and receives feedback in the form of rewards. Over time, it learns an optimal policy that maximizes cumulative reward. Over time, it learns a policy $\pi(a|s)$ that maximizes the expected cumulative reward:

$$\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right],$$

where $\gamma \in [0, 1)$ is a discount factor controlling the importance of future rewards.

2.2 Decision Tree Learning

Decision Trees (DT) are a fundamental subset of machine learning algorithms that provide a visual representation by recursively dividing the input feature space $\mathcal{X}$ into distinct, homogeneous regions based on simple decision rules. Thus, leading to visually interpretable paths that yield the model's outcome, represented by the leaf nodes.

2.2.1 Historical Development

The origins of decision tree learning can be traced to the early 1960s, when *Morgan and Sonquist* (1963) introduced the *Automatic Interaction Detection (AID)* algorithm [39]. AID was developed to uncover meaningful partitions in survey data by recursively splitting variables into subgroups that minimized within-group variance. Although simple in its construction, AID established the foundational principle of recursive binary partitioning that continues to underpin all subsequent tree-based models. Building on this concept, *Kass* (1980) proposed the *Chi-squared Automatic Interaction Detection (CHAID)* algorithm [40], which generalized AID to handle categorical predictors and multiway splits. CHAID used statistical tests of independence, such as the chi-squared test, to determine the best partitioning variables, thereby introducing a more formal statistical criterion for split selection.

A major methodological milestone occurred with the publication of *Classification and Regression Trees (CART)* by *Breiman, Friedman, Olshen, and Stone* (1984) [3]. CART unified classification and regression under a single framework and introduced well-defined impurity measures, such as the *Gini index* and variance reduction, as criteria for optimal splitting based on the importance of the features on the outcome. Importantly, it established the principle of *cost–complexity pruning*, which systematically removes subtrees that do not contribute to improved generalization. CART also formalized how to handle continuous attributes through threshold-based binary splits and how to manage missing data using surrogate splits—techniques that remain standard in modern implementations.

In parallel, *Quinlan* (1986) proposed the *Iterative Dichotomiser 3 (ID3)* algorithm [4], which adopted an information-theoretic approach to tree induction. ID3 selected attributes that maximized *information gain*, derived from the reduction in Shannon entropy after a split. This provided a probabilistic interpretation of decision boundaries and directly linked decision tree learning to principles of information theory. Quinlan later extended this work with *C4.5* (1993) [41], introducing mechanisms for handling continuous-valued features, missing values, and overfitting via post-pruning based on estimated error rates. C4.5 became one of the most widely used decision tree algorithms, influencing both academic research and practical applications.

Together, these developments, AID, CHAID, CART, ID3, and C4.5, transformed decision trees from heuristic segmentation tools into a robust family of supervised learning algorithms grounded in statistical theory. They established the conceptual and methodological basis for later advances such as oblique trees, ensemble methods, and, ultimately, optimal decision tree formulations.

2.2.2 Basics of Binary Tree Induction

Decision tree learning is *usually* treated as a supervised learning paradigm—this is the setting adopted here and by the algorithms discussed in the previous sections—where the aim is to approximate an unknown mapping $f : \mathcal{X} \to \mathcal{Y}$ from training data $\mathbb{D} = \{(x_i, y_i)\}_{i=1}^n$, with $x_i \in \mathcal{X} \subseteq \mathbb{R}^D$ and $y_i \in \mathcal{Y}$, where $\mathcal{X}$ indicates the input feature space and $\mathcal{Y}$ is the set of target labels. Unsupervised (and semi-supervised) variants of decision trees do exist, such as density estimation trees for modeling $p(\mathbf{x})$ and unsupervised clustering trees, but they are not the focus of our subsection [42, 43].

The model is chosen from a hypothesis class $\mathcal{F}$ to minimize the empirical prediction loss over the dataset, often regularized by a complexity term:

$$\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) + \lambda\, \Omega(f),$$

where $\ell(\cdot)$ measures the prediction error, and $\Omega(f)$ penalizes excessive model complexity [44, 45]. In the specific case of decision trees, f is represented by a hierarchical structure T that recursively partitions the input space into disjoint regions, assigning a constant prediction c_ℓ to each leaf region R_ℓ. Learning the tree corresponds to solving the optimization problem

$$\min_{T \in \mathcal{T}} \frac{1}{n} \sum_{i=1}^{n} \ell(f_T(x_i), y_i) + \alpha\, |T|,$$

where $|T|$ measures the tree size and α controls the trade-off between accuracy and simplicity [3]. This formal objective underlies both classical greedy methods and modern globally optimal approaches, connecting decision trees to the broader principles of statistical learning theory.

The core principle underlying binary decision trees is the recursive partitioning of the input feature space $\mathcal{X} \subseteq \mathbb{R}^D$ into a set of axis-aligned, disjoint cuboid regions[37], as visualized by Figure 2.2.1 where the partition is represented in two dimensions for simplicity. Each region is associated with a simple predictive model—typically a constant value in the case of *regression*, or a discrete class label for *classification*. Every region R_i corresponds to a terminal (leaf) node l_i, while the traversal from the root node to a leaf defines a unique sequence of binary decisions over the feature variables. In this way, the model learns a nonlinear mapping $f : \mathcal{X} \to \mathcal{Y}$. This mapping is constructed as a composition of region-specific local decision rules.

Formally, the model can be expressed as

$$f_T(x) = \sum_{\ell=1}^{L} c_\ell\, \mathbb{1}\{x \in R_\ell\}, \tag{2.2.1}$$

where $\mathbb{1}_{\{\cdot\}}$ denotes the indicator function and c_ℓ the prediction assigned to region R_ℓ.

The prediction rules for regression tasks correspond to the mean target value within the region

$$c_\ell = \frac{1}{|R_\ell|} \sum_{\mathbf{x}_i \in R_\ell} y_i,$$

while for classification, they are determined by the majority class,

$$c_\ell = \arg\max_k p_k(R_\ell),$$

where $p_k(R_\ell)$ represents the empirical class proportion of class k within R_ℓ [3, 37].

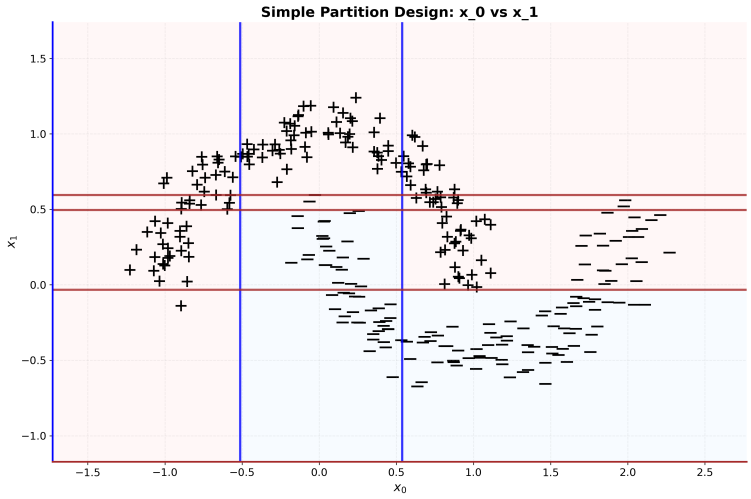

Figure 2.2.1: Partition of feature space $\mathcal{X}$ into cuboids in 2D.

Instances are marked with '+' or '–' to indicate their class (positive or negative.

Each internal node t applies a decision rule $a_t^\top x_i \leq b_t$, where $a_t \in \mathbb{R}^p$ and $b_t \in \mathbb{R}$ denote the split coefficients and threshold, respectively, which split the data into left R_L and right R_R children subsets. This process of *recursive partitioning* continues independently in each child node until a specific termination criterion is reached, such as minimum samples per node, maximum allowed depth. The final result is a visually interpretable structure that ensures the hierarchical importance of features and the local final outcomes assigned to leaves, depicted by Figure 2.2.2.

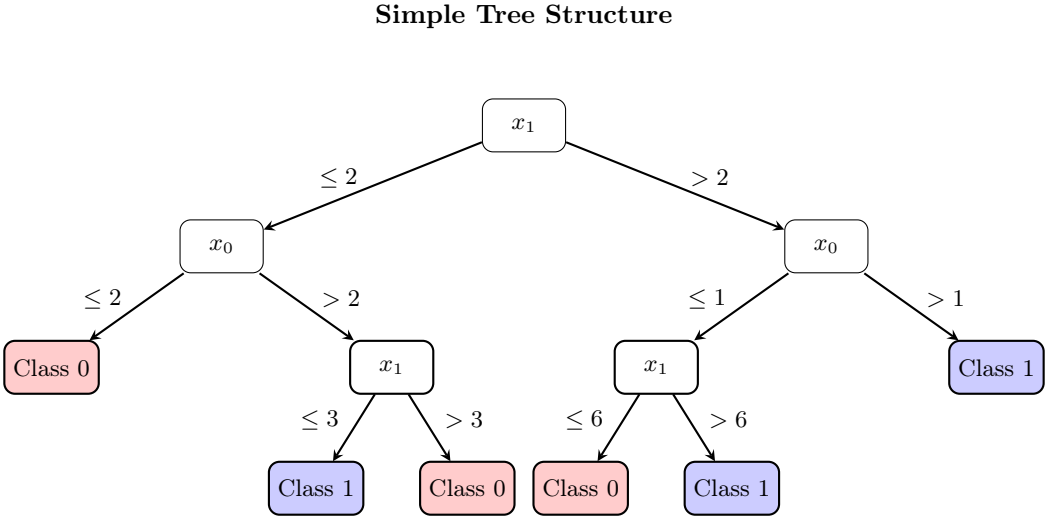

Figure 2.2.2: Balanced decision tree with non-overlapping leaves.

The layout ensures that all leaf nodes are evenly distributed along the bottom. Each internal node represents a threshold condition, and the branches terminate cleanly in class predictions. Blue leaves denote *Class 1*, and red leaves denote *Class 0*.

The criterion for selecting splits is centered around maximizing the reduction in an *impurity* measure $I(t)$ that quantifies the heterogeneity of a node t. Intuitively, impurity quantifies how mixed the class labels are among the samples that reach a particular node, meaning the higher the variety of classes that reach the specific node, the higher the impurity measure. The goal of each split is to ensure the impurity of each child node is lower than its parent, thus partitioning the space into more homogeneous and specific subsets with

respect to the label variable y.

For classification tasks, the most widely used impurity metrics are the *Gini index* and the *entropy*:

$$I_{\text{Gini}}(t) = \sum_{k=1}^{K} p_k(1 - p_k), \tag{2.2.2}$$

$$I_{\text{Entropy}}(t) = -\sum_{k=1}^{K} p_k \log p_k, \tag{2.2.3}$$

where p_k denotes the empirical proportion of samples of class k within node t. For regression, node impurity is typically defined as the residual variance or mean squared error:

$$I_{\text{Var}}(t) = \frac{1}{N_t} \sum_{i:\mathbf{x}_i \in t} (y_i - \bar{y}_t)^2, \quad \bar{y}_t = \frac{1}{N_t} \sum_{i:\mathbf{x}_i \in t} y_i, \tag{2.2.4}$$

where y_i denotes the target value of the i-th training instance that reaches node t, and $\bar{y}_t$ represents the mean target value of all samples that reach the node.

The optimal split (x_j, θ) at node t is chosen to maximize the impurity reduction:

$$\Delta I = I(t) - \sum_{j \in \text{children}(t)} \frac{N_j}{N_t} I(j), \tag{2.2.5}$$

where N_t and N_j denote the number of instances in the parent and child nodes, respectively [3, 4, 41].

The greedy algorithm follows the *Top-Down Induction* approach[4], the procedure of which is described below:

- Start at root node t with all data.
- Evaluate all possible features x_i and thresholds θ
- Select the split on the feature that maximizes the impurity reduction ΔI
- Recursively apply the process on child nodes
- Terminate based on the stopping criterion

While the greedy procedure enables computationally efficient training, it does not ensure a globally optimal tree. Joint optimization of split variables, thresholds, and tree structure constitutes a combinatorial problem that is NP-complete [7]. As a result, trees generated by recursive partitioning can vary substantially due to minor changes in the data or initial conditions, highlighting the non-convexity of the search space. This challenge has led to research on pruning, regularization, and globally optimal tree induction methods, which balance accuracy and model complexity to enhance generalization performance [3, 45].

2.2.3 Pruning, Regularization, and Generalization

The challenge of most substantial importance in decision tree learning has been the control of *overfitting*, as baseline algorithms tend to create complex models (greater depth and thus more leaf nodes), which usually lose accuracy on test scenarios. *Pruning* is the primary mechanism introduced as a solution for the simplification of the tree structure after the growth has reached a certain point, thus improving generalization. This idea dates to the CART[3] and C4.5[41], indicating that a grown tree often fits redundant instances referred to as noise or spurious patterns that are unique on the training set. The removal of such sub-trees that fit on these special cases leads to improved test performance, even though training accuracy is reduced [46]. Practically, this result addresses the bias-variance tradeoff, where a smaller tree has higher bias, but the variance is substantially lower[45]. We note that the regularization term is not universally fixed to tree size, as while CART penalizes $|T|$, modern optimal-tree methods such as in Optimal Decision Tree applications [2, 47], employ more general $\Omega(T)$ penalties, including sparsity-aware and structure-aware regularizations.

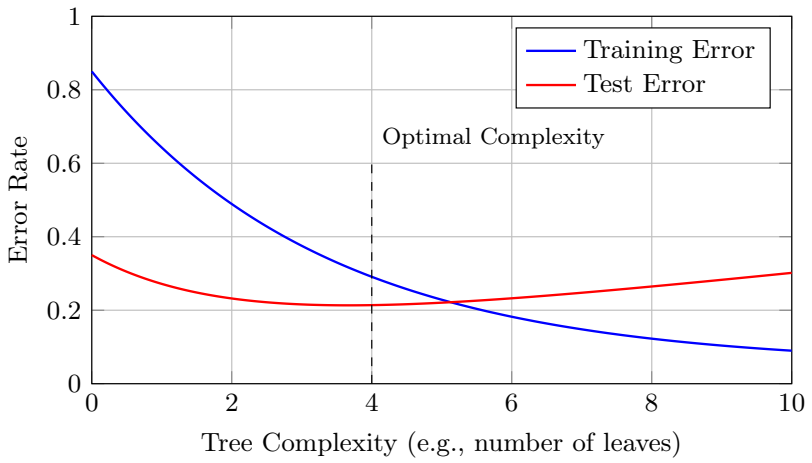

Figure 2.2.3: Effect of pruning on training and test error.

Pruning selects the tree complexity that minimizes validation error by balancing bias and variance. The training error decreases monotonically with complexity, while the test error forms a U-shaped curve, indicating overfitting beyond the optimal point [3, 45].

Pre-pruning Strategies. Pre-pruning (early stopping) limits tree growth before it perfectly fits the training data, using criteria such as maximum depth, minimum samples per node, or minimum information gain. These constraints act as regularizers that prevent overly specific splits, though excessively strict limits can cause underfitting [41].

Post-pruning Algorithms. A more robust approach is to first grow a full tree and then prune subtrees that do not improve validation performance. *CART* introduced cost–complexity pruning, minimizing

$$\text{Error}(T) + \lambda \cdot \boldsymbol{\Omega(T)},$$

where λ penalizes tree size. Cross-validation and the one–standard–error rule are used to select the optimal λ [45]. *C4.5*, in contrast, applies error-based pruning based on statistical confidence intervals: if a subtree's estimated error is not significantly lower than a leaf's, it is replaced.

Impact on Generalization. Pre-pruning and post-pruning techniques seek to optimize accuracy and model simplicity by eliminating branches that lack significant predictive value. Pruning primarily functions as a variance reduction mechanism. By removing sub-trees that capture random noise instead of the underlying data structure, it limits model flexibility and yields more consistent predictions across data samples. Empirical analyses consistently show that pruned trees achieve lower test error than fully grown ones, even at the cost of slightly higher training error, confirming that many fine-grained splits reflect noise rather than informative patterns. This directly relates to the classical *bias–variance trade-off*, where unpruned trees exhibit low bias but high variance. In contrast, pruned trees sacrifice some fit to gain robustness and improved generalization performance. In practical applications, the optimal degree of pruning is typically determined using cross-validation or validation-set tuning to minimize predictive error on previously unseen data.

In addition to mitigating overfitting, pruning improves both *interpretability* and model *stability*. Simpler trees are easier to visualize, validate, and explain, particularly in high-stakes decision-making domains such as medicine, finance, and policy analysis. By retaining only statistically significant splits, pruning ensures that the resulting structure emphasizes the most relevant relationships in the data. Consequently, pruning and related regularization methods, such as depth constraints, penalty-based formulations, or ensemble averaging, remain essential for building decision trees that generalize well while preserving human interpretability [45]. This pursuit of balancing performance and transparency has also inspired later advances in ensemble approaches, where multiple trees are combined to further stabilize predictions and enhance accuracy.

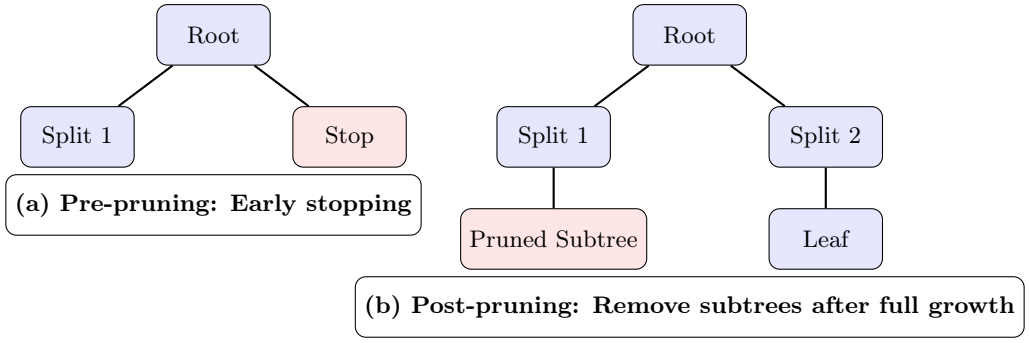

Figure 2.2.4: Comparison of pre-pruning and post-pruning strategies.

Pre-pruning halts tree growth based on heuristics such as minimum information gain or sample size, while post-pruning removes subtrees that fail to improve validation performance after the tree is fully grown. Post-pruning generally yields more reliable models with improved generalization [35].

2.2.4 Recent Advances and State-of-the-art models

Since the 1990s, research on decision trees has yielded methodological and algorithmic breakthroughs that have significantly improved predictive accuracy and computational scalability. While the classical decision tree remains fundamental to interpretable machine learning, contemporary variants have advanced beyond the traditional greedy-split paradigm. This section reviews key areas of progress in tree-based methods: ensemble learning, oblique and hybrid tree models, and optimization-based formulations, which define the current state-of-the-art.

Ensemble Methods.

The most influential advances in decision tree research stem from the introduction of *ensemble methods*, which combine multiple trees to form a single, stronger predictive model. The fundamental idea is to reduce the high variance inherent in individual trees by averaging or aggregating across many of them. *Bagging Bootstrap Aggregating*[48] is the earliest such approach, where each tree is trained on a bootstrap-resampled version of the data and the ensemble's prediction is obtained through averaging (for regression) or majority voting (for classification). *Random Forests* [49] extended this principle by introducing random feature selection at each split, thereby decorrelating the constituent trees and further improving generalization. Random Forests remain among the most widely used models for tabular data due to their robustness, resistance to overfitting, and ease of parallelization.

A complementary approach, *boosting*, constructs ensembles in a sequential manner: each new tree focuses on correcting the errors of the previously learned ones. The seminal *AdaBoost* algorithm [50] demonstrated that combining many weak learners, often very shallow trees, can yield a strong classifier with low training and generalization error. Later, *Gradient Boosting* [51] unified boosting with gradient-based optimization by fitting each successive tree to the negative gradient of a differentiable loss function. Modern implementations such as *XGBoost* [52] and *LightGBM* [30] introduced additional improvements, including regularization terms, histogram-based split finding, and distributed training capabilities. These boosting frameworks have achieved top performance on canonical structured-data benchmarks such as the *UCI Adult Income* dataset [33] and the *German Credit (Statlog)* dataset [31], which remain standard baselines for evaluating generalization and robustness on tabular data. However, while ensembles significantly improve predictive performance, they do so at the cost of interpretability: the combined effect of hundreds of small trees is difficult to visualize or explain in a human-understandable form.

Oblique and Hybrid Decision Trees.

In parallel to ensemble research, another line of work sought to enhance the representational power of single trees by generalizing the type of decision boundaries they can form. Traditional trees perform *axis-aligned* splits, where each internal node tests a threshold on a single feature (e.g., $x_j < \theta$). *Oblique decision trees*, in contrast, allow splits based on linear combinations of features, corresponding to hyperplane partitions of

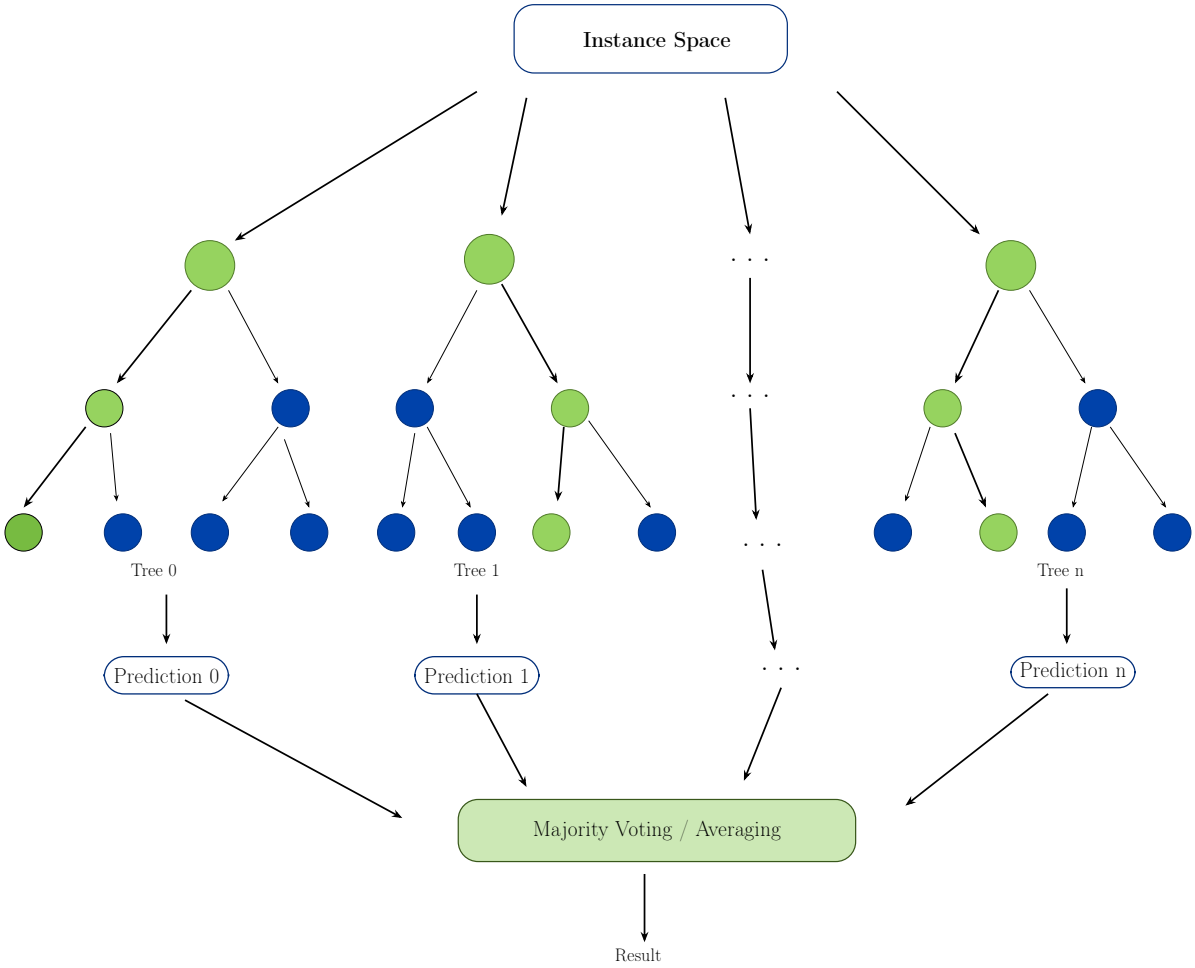

Figure 2.2.5: Tree ensembles example - Random Forest visualized

the input space. The OC1 algorithm[53] is an early example of this family, using randomized and gradient-based methods to optimize the coefficients of linear tests at each node. Such oblique splits can produce more compact trees by replacing several axis-aligned splits with a single, multi-feature condition. Nonetheless, they introduce additional computational complexity and reduce interpretability, as linear combinations (e.g., $3.7x_1 + 1.2x_2 < 5.4$) are less intuitive than univariate thresholds.

Beyond oblique models, *hybrid trees* integrate tree-based structures with other machine learning paradigms to combine their complementary strengths. For instance, trees with linear models in their leaves (model trees) can capture local linear relationships within regions, improving accuracy while maintaining some interpretability. Recent developments in *differentiable decision trees* and *neural decision forests* extend this idea into the deep learning domain: by using soft (probabilistic) splits and backpropagation, researchers have trained tree-like architectures end-to-end alongside neural networks. These methods bridge the gap between the clear, rule-based reasoning of trees and the complex pattern-learning ability of deep models[54].

Trade-offs, Accuracy, and Interpretability.

Across all these advances, a recurring theme is the trade-off between predictive performance and interpretability. Ensembles such as Random Forests and Gradient Boosted Trees deliver state-of-the-art accuracy on a wide range of structured prediction tasks, often surpassing single-tree methods by a large margin. However, they forfeit the clear decision logic that makes single trees appealing in high-stakes domains like healthcare, finance, and policy. The growing interest in Explainable Artificial Intelligence (XAI) has therefore revived attention to single-tree methods that can remain competitive with ensembles without sacrificing transparency.

To address this, recent research, such as the *Optimal Classification Trees* (OCT) framework proposed by [5], has recast tree learning as a global optimization problem. By formulating the split-selection process as a mixed-integer optimization task rather than a greedy heuristic, these models achieve near-optimal trade-offs between complexity and accuracy. They effectively close part of the performance gap between interpretable single trees and opaque ensembles. Such work bridges traditional decision trees with modern optimization and continues to shape the frontier of interpretable machine learning.

2.3 Optimal Decision Trees

Learning an optimal decision tree is a well-known NP-complete problem in its decision form, and NP-hard in its optimization form [7]. The result showed that even when the goal is to minimize the expected number of tests (path length), thus the minimal decision tree, required to identify an object (the dataset), no polynomial-time algorithm exists unless $P = NP$. Specifically, they proved the minimum decision tree problem is NP-complete by reduction from the Set Cover problem variant referred to as *EC3*, which describes the Exact Set Cover, where each of the subsets available contains exactly three elements. Formally, given a finite set of training instances $\mathbb{D} = \{(x_i, y_i)\}_{i=1}^{n}$ with $x_i \in \mathbb{R}^d$, the goal is to find a tree T minimizing a cost function $\mathcal{L}(T)$ (e.g., the misclassification rate or the number of internal nodes), subject to perfect classification of the training samples:

$$\min_{T \in \mathcal{T}} \mathcal{L}(T) \quad \text{s.t.} \quad f_T(x_i) = y_i, \ \ \forall i \in \{1, \ldots, n\},$$

where $\mathcal{T}$ is the set of all possible trees under consideration. It was shown that even deciding whether there exists a tree of size at most k consistent with the data is NP-complete. This is because the number of distinct trees of depth d grows super-exponentially with both d and the number of features p:

$$|\mathcal{T}(p, d)| = O\Big((2p)^{2^d - 1}\Big), \tag{2.3.1}$$

implying that an exhaustive search over all possible structures is computationally infeasible even for moderate values of p and d.

Due to this intractability, early algorithms such as CART [3] and ID3 [4] relied on greedy, top-down heuristics that make locally optimal splitting decisions. However, advances in optimization algorithms, hardware, and solver technology have made exact methods increasingly tractable. It is noted by [5] that modern mixed-integer optimization (MIO) solvers have improved by nearly 10^{11}–10^{12} times over the past few decades, enabling globally optimal tree construction for moderate-sized datasets.

Three main paradigms now dominate exact tree-learning research:

1. **Mixed-Integer Optimization (MIO)** formulations that encode tree construction as a single mathematical program;
2. **Constraint Programming (CP)** and **SAT**-based formulations that rely on combinatorial search with pruning and caching; and
3. **Dynamic Programming (DP)** frameworks that exploit the decision tree problem solutions, by recursively solving optimal sub-solutions that gradually compose larger global optimal solutions, based on the foundations of DP algorithms and the recursive equation objective of the task. In addition, they employ *smart bounding mechanisms* to drastically reduce the search space, such as the *similar-support bound* introduced by [14], which prunes redundant subproblems that share overlapping instance sets.

2.3.1 Mixed-Interger Optimization Programming

The first family of exact algorithms to achieve global optimality formulates decision-making tree learning as a *mixed-integer optimization (MIO)* problem. This approach, introduced by [5], constructs the entire tree structure and its prediction assignments simultaneously within a single mathematical formulation. Unlike the greedy heuristics, which commit to one split at a time, the MIO mechanism searches the combinatorial space of all possible splits, thresholds, and leaf labels in a globally coordinated manner.

To present the problem, we revise the setting of Binary Decision Tree from Section 2.2.2, where we consider the dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, with $x_i \in \{1, \ldots, K\}$. We consider a full binary tree of depth D containing $|T| = 2^D - 1$ internal node and 2^D leaves, and the branch rules described by

$$a_t^\top x_i \leq b_t,$$

where $a_i \in \mathbb{R}$ and $b_i \in \mathbb{R}$ denote the split coefficients and thresholds, respectively. When splits are restricted to axis-aligned tests, $a_i 2$ is constrained to be a unit vector in the standard basis, i.3., $a_t = e_j$ for some feature j. The resulting model partitions the feature space X into disjoint regions using the prediction function 2.2.1.

The MIO formulation introduces binary and continuous variables to represent the structure and predictions of the tree. For each branch node t, binary variables a_{jt} indicate which feature j is selected for the split, and b_t denotes the corresponding threshold. For each training instance i and leaf ℓ, $z_{i\ell}$ specifies whether sample i is assigned to leaf ℓ, and $c_{\ell k}$ denotes whether leaf ℓ predicts class k. Finally, L_ℓ indicates whether a leaf is active. Routing constraints ensure that each instance reaches exactly one leaf:

$$\sum_{\ell=1}^{L} z_{i\ell} = 1, \qquad i = 1, \ldots, n.$$

To guarantee consistency with the tree structure, a sample can be assigned to a leaf only if it satisfies all branch inequalities along its corresponding path. If $\mathcal{A}_\ell^L$ and $\mathcal{A}_\ell^R$ denote the sets of branch nodes where the sample must go left or right, respectively, these conditions are expressed as

$$a_t^\top x_i \leq b_t + M(1 - z_{i\ell}), \quad \forall t \in \mathcal{A}_\ell^L,$$
$$a_t^\top x_i \geq b_t + \varepsilon - M(1 - z_{i\ell}), \quad \forall t \in \mathcal{A}_\ell^R,$$

where M is a sufficiently large constant and $\varepsilon > 0$ enforces strict separation. Each leaf predicts a single class, enforced by

$$\sum_{k=1}^{K} c_{\ell k} = L_\ell, \qquad z_{i\ell} \leq L_\ell, \qquad c_{\ell k} \in \{0, 1\}.$$

The objective function jointly minimizes empirical misclassification error and a sparsity-inducing penalty that controls model complexity:

$$\min_{a,b,z,c,L} \; \frac{1}{n} \sum_{i=1}^{n} \sum_{\ell=1}^{L} \sum_{k=1}^{K} \mathbb{1}\{y_i \neq k\}\, z_{i\ell} c_{\ell k} + \alpha \sum_{\ell=1}^{L} L_\ell. \tag{2.3.2}$$

The first term measures the fraction of misclassified samples, while the second penalizes the number of active leaves, promoting interpretability. The parameter $\alpha > 0$ serves the same role as the cost–complexity pruning constant in CART, balancing accuracy and sparsity.

Putting these elements together, the complete *Optimal Classification Tree (OCT)* model is written as:

$$\begin{aligned}
\min_{a,b,z,c,L} \quad & \frac{1}{n} \sum_{i=1}^{n} \sum_{\ell=1}^{L} \sum_{k=1}^{K} \mathbb{1}\{y_i \neq k\}\, z_{i\ell} c_{\ell k} + \alpha \sum_{\ell=1}^{L} L_\ell \\
\text{s.t.} \quad & \sum_{j=1}^{p} a_{jt} = 1, \quad a_{jt} \in \{0, 1\}, \quad b_t \in \mathbb{R}, \quad t = 1, \ldots, T, \\
& \sum_{\ell=1}^{L} z_{i\ell} = 1, \quad z_{i\ell} \in \{0, 1\}, \quad i = 1, \ldots, n, \\
& a_t^\top x_i \leq b_t + M(1 - z_{i\ell}), \quad \forall t \in \mathcal{A}_\ell^L, \\
& a_t^\top x_i \geq b_t + \varepsilon - M(1 - z_{i\ell}), \quad \forall t \in \mathcal{A}_\ell^R, \\
& \sum_{k=1}^{K} c_{\ell k} = L_\ell, \quad c_{\ell k}, L_\ell \in \{0, 1\}.
\end{aligned} \tag{2.3.3}$$

The product $z_{i\ell}c_{\ell k}$ introduces nonlinearity, making the formulation a mixed-integer nonlinear program (MINLP). However, this can be linearized using auxiliary variables, yielding a mixed-integer linear program (MILP) that can be solved with modern solvers. When the splits are restricted to one feature per node, the feasible region forms a polyhedral set, allowing the use of efficient branch-and-bound procedures[1] to find the global optimum.

The model simultaneously optimizes over all splitting features, thresholds, and leaf assignments, ensuring that routing and labeling are consistent. It provides a globally optimal tree with respect to the specified depth and objective function, achieving a principled trade-off between interpretability and predictive accuracy. It is stated by [5] that optimal trees of depth up to four can be computed for datasets containing several thousand instances, often surpassing CART in accuracy while remaining equally transparent.

Subsequent studies extended the MIO approach to improve scalability and numerical stability. The *Binary Optimal Classification Tree* (BinOCT) was introduced by [8], a binary linear programming variant that decouples the number of constraints from the dataset size. Instead of representing every instance explicitly, BinOCT encodes feasible decision paths with binary vectors corresponding to node activations, reducing the formulation from $\mathcal{O}(n2^D)$ to $\mathcal{O}(2^D)$ constraints. This compactness significantly reduces computational overhead and memory usage, enabling depth-4 optimal trees for datasets with tens of thousands of points.

FlowOCT was later proposed by [9], which reformulates the routing process as a network-flow problem. Each sample "flows" from the root to one leaf through binary variables representing left or right decisions at each branch. Flow-conservation constraints ensure that every instance is assigned to exactly one leaf, while eliminating symmetric subtrees and tightening the LP relaxation. This leads to faster convergence and smaller integrality gaps, allowing the model to scale more efficiently in both depth and sample size. Further improvements were made by [57], where the resulting trees demonstrate better generalization performance.

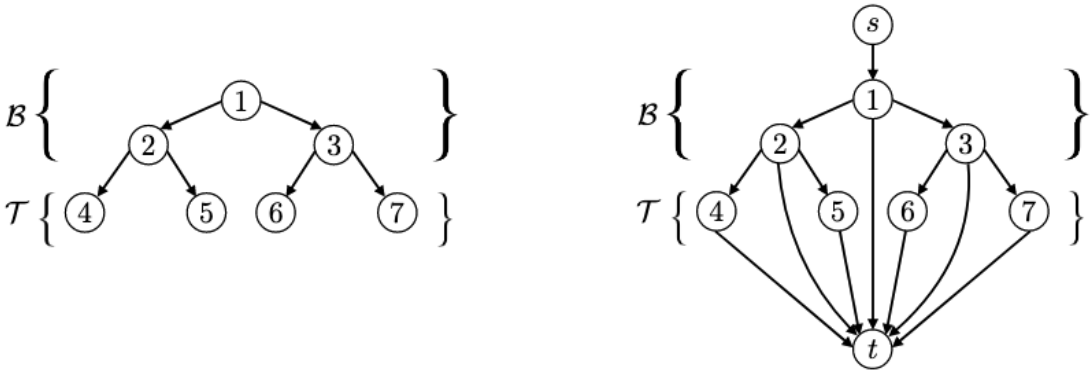

Figure 2.3.1: Representation of flowOCT

A decision tree (left) and its associated flow graph (right). B, T indicate the branching and leaf nodes, respectively. Reproduced from[9].

Empirical comparisons demonstrate that FlowOCT yields the most consistent results across datasets, whereas BinOCT remains the most scalable for large sample sizes.

The flexibility of the MIO framework also allows the construction of *oblique* or multivariate decision trees. In the *Optimal Classification Tree with Hyperplanes* (OCT-H) [5], the decision rule at each node takes the form $w_t^\top x_i \le b_t$, with continuous coefficients $w_t \in \mathbb{R}^p$. The one-hot restriction on a_{jt} is replaced by the constraint $\|w_t\|_1 \le 1$, and a sparsity-inducing penalty is added to control model complexity:

$$\min_{w,b,z,c,L} \ \frac{1}{n}\sum_{i=1}^{n}\sum_{\ell,k} \mathbb{1}\{y_i \neq k\}\, z_{i\ell}c_{\ell k} + \alpha \sum_{\ell} L_\ell + \lambda \sum_{t=1}^{T} \|w_t\|_1. \tag{2.3.4}$$

[1] Branch-and-Bound is a general global optimization strategy introduced by [55], and later formalized in [56].

The ℓ_1-regularization term penalizes dense hyperplanes and promotes sparsity, ensuring interpretability similar to Lasso regression. This multivariate extension produces more expressive decision boundaries and typically achieves 4–7% higher accuracy on high-dimensional datasets, though at a small cost in transparency.

Overall, MIO-based formulations have demonstrated that globally optimal and interpretable decision trees can be derived using modern optimization solvers. Models such as OCT, BinOCT, FlowOCT, and OCT-H demonstrate that joint optimization of structure and prediction yields consistent performance gains and a clear cost–complexity trade-off. Nevertheless, the number of feasible tree structures grows super-exponentially with depth (eq. 2.3.1) and even with efficient pruning and parallelization, MIO approaches remain computationally demanding for very large datasets. These challenges have motivated alternative paradigms, most notably constraint programming, SAT formulations, and dynamic programming approaches, which preserve global optimality while offering greater scalability for deeper or larger-scale problems.

2.3.2 Constraint and SAT-Based Formulations

A second class of approaches for the optimal tree induction relies on *Constraint Programming (CP)* and *Boolean Satisfiability (SAT)* formulations. These methods encode the tree structure, instance routing, and labeling decisions as collections of discrete constraints, solved using propagation and pruning within constraint solvers. Unlike MIO/MILP, which operates on continuous relaxations of integer variables, SAT-based and CP-based formulations are entirely symbolic, reasoning directly over Boolean variables through logical inference. This makes them particularly well matched for datasets with discrete features of finite sets of threshold sets.

At their core, SAT formulations aim to decide whether there exists an exact assignment of such Boolean variables that satisfies a propositional formula. Formally, the SAT [58] problem can be stated as:

Given a Boolean formula $\Phi(X_1, X_2, \ldots, X_n)$ in conjunctive normal form (CNF), where each X_i denotes a Boolean variable, the SAT problem determines whether

$$\exists\,(x_1, \ldots, x_n) \in \{0,1\}^n \text{ such that } \Phi(x_1, x_2, \ldots, x_n) = \text{True}.$$

In context of decision tree learning, the propositional formula Φ encodes the logical structure of a tree of fixed depth D. Each node decision, branch routing, and leaf assignment is represented by a Boolean variable. For instance, variables can indicate whether node t splits in feature j, whether sample i reaches leaf l, or leaf l predicts a particular class label. A satisfying assignment to these variables thus defines a valid tree consistent with the data.

The first explicit encoding of decision tree learning as a SAT instance was proposed by [10], who demonstrated that compact CNF representations can yield provably minimal trees for moderate-size datasets (order of magnitude of $n \in [100, 10^3]$ instances, $p \in [5, 50]$ features, with depth $D \leq 5$). The central idea is to constrain each training instance to be assigned to exactly one leaf, and each leaf to correspond to exactly one label. These logical relationships can be formalized as:

$$\bigvee_{\ell=1}^{2^D} z_{i\ell}, \qquad \neg z_{i\ell} \vee \neg z_{i\ell'},\ \forall \ell \neq \ell', \tag{2.3.5}$$

where $z_{i\ell}$ is a Boolean variable indicating whether instance i reaches leaf ℓ. The first clause ensures that every instance must be assigned to *at least one* leaf, while the second set of clauses enforces the exclusivity, that is, exactly one leaf, meaning that for each internal node t, a training instance can proceed left or right only of its feature values satisfy the corresponding branch conditions. These conditions can be encoded as Horn clauses [59, 60] that connect node-selection and routing variables:

$$(\text{split}_t^j \wedge b_{ij}^{(\leq \theta_t)}) \rightarrow \text{left}_t(i), \qquad (\text{split}_t^j \wedge \neg b_{ij}^{(\leq \theta_t)}) \rightarrow \text{right}_t(i).$$

Collectively, these constraints define a satisfiability instance whose feasible solutions correspond to a valid decision tree of depth D consistent with all training samples.

The objective of the SAT-based formulation is to find the *smallest consistent tree*, that is, the minimal D for which the SAT problem admits a satisfying assignment. This is achieved by iteratively increasing D

and resolving until feasibility is attained. The resulting model is guaranteed to have the minimal number of levels (depth) achieving perfect classification accuracy, which is analogous to minimizing the empirical error under a size constraint in MIO formulations, but solved through combinatorial search. In practice, the search process exploits clause learning and unit propagation to prune infeasible partial trees (subsolutions), making significantly more efficient than naive enumeration.

CP generalizes this logic-based paradigm to support numerical thresholds and categorical attributes of higher cardinality. Instead of pure Boolean clauses, CP uses structured constraint types such as *AllDifferent*, *AtMostOne*, and *reified linear constraints*, which in return capture higher-level relationships between decision variables. The CP and MaxSAT formulations developed in [8, 11] combine linear and logical constraints, solving them via propagation and **branch-and-bound**. These mechanisms strengthen constraint propagation, drastically reducing the effective search and improving convergence rates.

Empirical observations indicate that SAT-based and CP-based solvers perform efficiently on discrete datasets of limited size, where the number of admissible thresholds and feature combinations remains small. Within these conditions, compact constraint encodings and logical propagation manage to retrieve optimal trees of shallow depth, typically up to $D = 3 - 5$, in reasonable time. However, when the data dimension increases or when continuous attributes require multiple threshold variables, both CP and MIO formulations suffer from a rapid increase of constraints and solver branching. Their runtime grows exponentially with each additional admissible split, and their scalability becomes restrictive for moderate to large datasets.

In contrast, the Dynamic Programming (DP) approach formulations demonstrate more stable behavior regarding the computational and scalable aspects. This is justified through the recursive solving of all possible partitions of the examined dataset, thus extracting all possible optimal sub-solutions and then resolving to combine those into larger partitions of the population, up to the global optimum. For this reason, while the CP and MIO approaches are equally useful and robust approaches for small and discrete datasets, they cannot scale easily, and most importantly, there exists no implementation that allows users to create diverse optimization tasks that formulate an objective through bi-optimization Pareto front.

2.3.3 Dynamic Programming

Dynamic programming (DP) provides a third paradigm for optimal decision tree learning, exploiting the recursive structure of trees to decompose the global optimization problem into independent subproblems. In contrast to MILP or SAT formulations that search the full combinatorial space directly, DP methods leverage the fact that the optimal solution for a given subset of data can be obtained from the optimal solutions of its child partitions. The divide-and-conquer principle is particularly effective when the objective function is **separable**, meaning the total objective value can be expressed as a sum of contributions from subtrees. Formally, one can define recurrence for the optimal value function. Let $\mathcal{S} \subseteq \mathcal{D}$ represent the subset of training instances reaching a node and let d denote the remaining depth available. The optimal cost $C(S, d)$ of building a subtree on S with depth at most d satisfies:

$$C(S,d) = \begin{cases} \min_{y \in \mathcal{Y}} \operatorname{Err}(S, y), & d = 0, \\ \min_{i,\theta} \left[C(S_L(i,\theta), d-1) + C(S_R(i,\theta), d-1) \right], & d > 0, \end{cases} \tag{2.3.6}$$

where $\operatorname{Err}(S, y)$ is the classification error if all points in S are assigned label y_i, and $S_L(J, \theta)$ and $S_R(j, \theta)$ are the left and right subsets induced by splitting feature x_i and threshold θ. This recurrence expresses the total cost of the best split as the sum of the optimal costs of its two child nodes, enforcing global optimality through local optimality of substructures. Essentially, *if a tree is optimal, then every subtree it contains is optimal* as well, a property that directly derives from the basic DP principle. Below we delve into the most substantial methods of this approach.

Early DP Algorithms for optimal trees

The first dynamic programming (DP) algorithms for optimal decision trees, most notably the *DL8* algorithm by [12], demonstrated that globally optimal binary trees could be efficiently constructed for categorical features by reusing solutions to previously solved subproblems. DL8 formulates tree induction as a search over an *itemset lattice*, where each decision path corresponds to an itemset, that is, a conjunction of feature

tests, and employs a caching table to store the optimal subtree solution for every encountered subset of training instances. This reuse of cached solutions eliminates redundant computations when identical subsets are reached via different split sequences, thereby reducing exponential redundancy inherent in brute-force enumeration.

Building upon this foundation, *DL8.5* was proposed by [13], which significantly advanced the scalability of exact decision tree induction through more sophisticated caching and bounding techniques. Specifically, DL8.5 introduced an *upper-bounding strategy* that, once the optimal cost of a branch is computed, restricts the maximum admissible cost for its sibling branch, pruning any splits that cannot surpass the current best solution. In parallel, a *lower bound on misclassifications* is maintained for partially constructed subtrees, enabling large regions of the search space to be excluded without explicit evaluation. These bounding mechanisms are integrated within a depth-first branch-and-bound scheme, while the underlying DP engine ensures that optimal substructures are cached and reused across branches. Through this synergy of pruning and memoization, DL8.5 achieves orders-of-magnitude speed-ups compared to earlier exact methods, including MILP- and SAT-based solvers, by exploiting the intrinsic combinatorial structure of the decision tree optimization problem. Empirical studies show that DL8.5 attains superior runtime performance and scalability on fully binary datasets while preserving global optimality guarantees.

Scaling with Bitset Caching: MurTree

A more recent advancement in the DP lineage is the *MurTree* algorithm proposed by [14], which extends the dynamic programming framework to handle larger datasets and continuous-valued features. MurTree introduces a highly efficient bitset-based data representation, encoding each subset of samples as a compressed bit vector that allows constant-time evaluation of candidate splits and extremely fast subset operations. This compact representation, combined with aggressive caching of intermediate optimal solutions, substantially reduces computational overhead and memory duplication. Furthermore, MurTree refines the bounding process by employing *similarity-based bounds*, which provide quick lower estimates on the number of misclassifications for any partial tree, enabling the algorithm to prune unpromising branches before full expansion. Together, these innovations allow MurTree to scale to datasets with tens of thousands of samples, achieving orders-of-magnitude runtime improvements over previous exact methods. Despite this efficiency, MurTree maintains full guarantees of global optimality for a specified maximum depth or node count, marking a major step toward making exact decision tree learning practical for real-world, large-scale problems.

Hybrid Dynamic Programming and Search: The *Branches* Algorithm

Recent advances combine dynamic programming with informed search strategies. The *Branches* algorithm by [15] formulates decision tree optimization as an *AND/OR graph search* problem, solved using the AO* search technique. Here, AND-nodes represent decision points where both child subtrees must be solved, by mirroring the additive cost structure of DP, while OR-nodes correspond to branching choices among alternative splits. Branches explicitly applies Bellman's optimality principle by defining a recursive state-value function, analogous to the DP value function, to compute the best achievable objective for each partial tree state. A best-first search guided by an admissible heuristic, the *Purification bound*, directs exploration toward the most promising branches while pruning regions of the graph that cannot yield an optimal solution. This design ensures both *anytime behavior*, providing feasible intermediate trees, and guaranteed global optimality upon termination. Despite being implemented in Python, Branches outperforms GOSDT in runtime benchmarks, pushing the scalability of exact tree learning even further through its hybrid use of heuristic guidance and DP-style caching.

Dynamic programming and its specialized successors share a central advantage: flexibility in defining and optimizing diverse objective functions. Since the optimization is explicitly global, these methods can be tailored to criteria beyond simple accuracy, while incorporating fairness penalties [61], optimizing for imbalanced data metrics like weighted error or AUC [47], or trading off accuracy and size for model sparsity [5]. Unlike greedy learners such as CART, which rely on local impurity measures (e.g., Gini or entropy) as surrogates, optimal formulations directly minimize the intended global criterion. Thus, a fairness-adjusted or cost-sensitive objective can be optimized exactly by simply redefining the loss function, making DP-based methods highly adaptable to domain-specific requirements.

Nevertheless, classical DP methods remain limited to objectives that are *additively separable* across subtrees. Most optimal tree formulations, whether DP- or MIO-based, assume linear, decomposable objectives (e.g., sums of per-leaf or per-instance terms). Non-linear or relational constraints, such as bounding the difference in positive prediction rates across groups, violate this separability and require more general frameworks. A common workaround is combining multiple goals into a single weighted sum, as in sparse or regularized objectives, but this assumes the composite remains additive. When objectives truly conflict, bi-objective or Pareto optimization becomes necessary, as [14], who enumerate trade-off frontiers (e.g., between accuracy and fairness). While such extensions broaden the expressive power of optimal trees, they also increase computational complexity, motivating the development of generalized frameworks that retain the optimal substructure property under multiple objectives.

In summary, dynamic programming constitutes a powerful and conceptually elegant framework for globally optimal decision tree induction. By caching and reusing optimal substructures, DP-based methods eliminate redundant computation and achieve a balance between optimality and efficiency that was long considered infeasible beyond toy examples. Over the past decade, a progression of increasingly sophisticated DP and DP-hybrid algorithms, spanning DL8 and DL8.5, MurTree, OSDT/GOSDT, and the more recent Branches and STreeD frameworks, has substantially expanded the attainable scales of optimal tree learning, from a few hundred instances to datasets containing hundreds of thousands, and from single-objective formulations to multi-criteria optimization. These developments demonstrate that exact optimization of decision trees, while inherently combinatorial, has become practical for real-world data sizes and complex performance trade-offs. Overall, dynamic programming and its modern extensions have become indispensable tools for tackling the optimal decision tree problem, enabling globally optimal and interpretable models at scales previously unreachable by exhaustive enumeration.

The next section focuses a work of [2], a generalization of the dynamic programming paradigm that establishes the theoretical conditions under which complex, multi-objective decision tree problems remain separable and optimizable within a recursive framework.

2.4 Related Work - Separable Trees with Dynamic Programming

The framework of *Separable Trees with Dynamic Programming (STreeD)* introduced by [2] was selected to be analyzed in a separate section as our formulation described in Section 4 is mainly built using the flexibility of the framework to design an optimization task tailored to the needs of any given objective context. Similarly to other ODT approaches that are created using DP formulation, STreeD exploits the independence of sub-tree solutions, preserving their optimality as sub-solutions. STreeD successfully achieves orders-of-magnitude better scalability than general-purpose solvers, while being able to adapt in a generalizable manner to any optimization task that is defined as separable under certain definitions and rules that the authors propose. The separability of a task, whether its purpose is to optimize one or multiple objectives, is defined and analyzed in this section as a mandatory basis to later define our own task via its constraints.

A key reason for selecting this framework is that alternative methods often exhibit limitations in computational efficiency or lack reproducible and adaptable frameworks. In particular, most Mixed Integer Programming (MIP), Constraint Programming (CP), and Satisfiability (SAT) approaches are well-defined but require extensive runtime even for moderate-sized datasets. Furthermore, both non-DP and current DP methods are limited in their ability to formulate diverse tasks, as they typically optimize only for accuracy or other non-linear objectives in isolation, thereby restricting adaptive reproducibility. In the section below, we will further discuss how the setting of an optimization tasks is defined under the framework, the advantage of global optimality of the DP approach, and analyze the use case of Pareto-optimal solutions of heterogeneous objectives of tasks that delve to solve multiple problems simultaneously in the form of one unified task.

2.4.1 Problem Setting

The problem setting of the STreeD formulation strictly tracks the logic of classic DP approaches for optimal decision trees, which recursively solve optimal sub-solutions(children) that additively combine their values into a parent optimal solution. To correctly define the distinctions of this current work, we need to align notation with the source representation, adopting the definition of the problem setting as presented by [2].

Let F be a finite set of binary features and K be the set of labels. A dataset is $D \subseteq \{0,1\}^{|F|} \times K$ with instances (x, k). The distinctive point of the dataset setting is that the features are purely binary, according to the given values $\{0, 1\}$ (also referred to as{True, False}), which requires the dataset to be preprocessed into a fully binarized format, a fundamental part of the problem correct implementation. Then, for any $f \in F$ let f and $\bar{f}$ denote the true or false value of a feature f of any instance of the dataset, and accordingly, we define the induced partitions

$$D_f \;=\; \{(x,k) \in D : x_f = 1\}, \qquad D_{\bar{f}} \;=\; \{(x,k) \in D : x_f = 0\},$$

as the sets of instances that for feature f have the assigned value True and False respectively.

A binary decision tree is $\tau = (B, L, b, l)$ with internal (branching) nodes B, leaf nodes L, a branching assignment $b : B \to F$, and a leaf labeling $l : L \to K$. Each $u \in B$ has left and right children u_L, u_R; instances satisfying the test on $b(u)$ are routed to the right child and the remainder to the left [2].

We consider as a canonical baseline objective, the misclassification problem. Writing $C(D, u)$ for the cost of the subtree rooted at u,

$$C(D,u) \;=\; \begin{cases} \displaystyle\sum_{(x,k)\in D} \mathbf{1}[k \neq l(u)]\,, & u \in L, \\ C\big(D_{b(u)},\, u_R\big) \;+\; C\big(D_{\overline{b(u)}},\, u_L\big), & u \in B. \end{cases} \tag{2.4.1}$$

Given a maximum depth d, the DP value recursive function that returns the minimum misclassification error is described by:

$$T(D,d) \;=\; \begin{cases} \min\{\,|D^+|,\; |D^-|\,\}, & d = 0, \\ \displaystyle\min_{f\in F}\Big\{\, T(D_f, d-1) \;+\; T(D_{\bar{f}}, d-1)\,\Big\}, & d > 0, \end{cases} \tag{2.4.2}$$

where D^+ and D^- denote the class-wise partitions induced by the target label. The recursion in (2.4.2) computes the minimum(optimal) objective value. In practical implementations, the application of memoization and bounding for efficiency is well met in this setting compared to methods of the same DP family. Similar to the authors, we adopt the notation of the distinction of terminology of *solution* to denote a tree solution τ and *value solution* to denote the value of the objective score of a given solution τ.

The generalization of this initial problem setting is achieved by defining the attributes of an *optimization task*. We define a state space $\mathcal{S}$ and a value space $\mathcal{V}$. The state space we refer to is the set $\mathcal{S}$of all problem configurations reachable during the tree search that fully determine the subproblem's data and constraints. For example, each node's dataset slice and path decisions. A solution space is the set $\mathcal{V}$ of all attainable objective values (or value tuples) that quantify the quality of candidate trees or sub-trees and on which comparison and combination operators are defined.

An optimization task is the tuple

$$o \;=\; \langle g,\; t,\; \succ,\; \oplus,\; c,\; s_0 \rangle,$$

whose components encode, respectively, local evaluation, state evolution, ordering, algebra of combination, feasibility, and the start state. The task components for a state $s \in \mathcal{S}$ and values in $\mathcal{V}$ are:

1. a cost function $g : \mathcal{S} \times (\mathcal{F} \cup \mathcal{K}) \to \mathcal{V}$ assigns the cost of taking action a in state s. Actions in the notation of the frameworks' basis[2] are either labeling a leaf with $\hat{k} \in \mathcal{K}$ or branching on $f \in \mathcal{F}$,
2. a transition function $t : \mathcal{S} \times \mathcal{F} \times \{0,1\} \to \mathcal{S}$ that returns the next state after branching on f to the right (1) or left (0),
3. a comparison operator $\succ$: $\mathcal{V} \times \mathcal{V} \to \{0,1\}$ which represents a strict comparison that determines when one value is strictly better than another in the context of Pareto dominance (e.g. $<$ for minimization),

[2]The notation is stated as of the original work of [2]. We make the distinction to avoid ambiguity in terminology with subsequent chapters that refer to actions as perturbation vectors of instances of a dataset, in the context of counterfactual explanations and actionable recourse.

4. a combining operator $\oplus : \mathcal{V} \times \mathcal{V} \to \mathcal{V}$ that combines the values of the two child sub-trees (and optionally aggregates a branching cost),
5. a constraint $c : \mathcal{V} \times \mathcal{S} \to \{0, 1\}$ is a feasibility predicate used to filter values that violate application constraints at state s.
6. and $s_0 \in \mathcal{S}$ is the initial state.

For the standard instantiation used throughout, the state records the current dataset and the set of features already used on the path, $s = \langle D, F \rangle$, the transition is

$$t\big(\langle D, F\rangle,\, f,\, 1\big) = \langle D_f,\, F \cup \{f\}\rangle, \qquad t\big(\langle D, F\rangle,\, f,\, 0\big) = \langle D_{\bar{f}},\, F \cup \{f\}\rangle, \tag{2.4.3}$$

and the initial state is $s_0 = \langle D, \emptyset\rangle$, as precisely given by the authors.

Given $o = \langle g, t, \succ, \oplus, c, s_0\rangle$, the value of a tree τ at node u in state s is computed recursively as

$$C(s, u) = \begin{cases} g(s,\, l(u))\,, & u \in L, \\ C\big(t(s, b(u), 1), u_R\big) \;\oplus\; C\big(t(s, b(u), 0), u_L\big) \;\oplus\; g(s,\, b(u))\,, & u \in B, \end{cases} \tag{2.4.4}$$

and the tree is evaluated from the root by $C(s_0, r)$. Equation (2.4.4) is the exact generalization of the classical "count-at-leaf, sum-over-children" recursion and fixes the interface by which $\{g, t, \oplus\}$ enter the dynamic program.

By returning to the canonical objective of misclassification, that is, with no ancillary costs, one chooses

$$g\big(\langle D, F\rangle, \hat{k}\big) = \big|\{(x, k) \in D :\; k \neq \hat{k}\}\big|, \qquad g\big(\langle D, F\rangle, f\big) = 0,$$

takes $\oplus$ to be addition, $\succ$ to be $<$, and $c \equiv 1$. Substituting these definitions into (2.4.4) yields the familiar recursive formulation for misclassification that underpins traditional dynamic programming methods, which are now expressed within a single, unified optimization framework.

The frameworks already implemented tasks included optimization objectives like cost–sensitive classification and prescriptive policy. In cost–sensitive classification, the leaf and branching evaluations incorporate heterogeneous penalties for different error types as well as optional measurement costs for acquiring features, so the learned tree trades accuracy against acquisition burden in a principled way. This instantiation preserves the same state and recursion as the basic setting while simply modifying how local decisions are scored, allowing practical deployment when errors or tests have unequal consequences.

In prescriptive policy learning, each leaf issues a concrete recommendation (treatment), and we score that choice by its expected benefit, so the tree serves as a clear, rules-based policy rather than a standard classifier. The same dynamic program applies without modification, branch costs can be added if needed, and practical limits such as budgets or capacities are handled as simple feasibility constraints without changing the overall structure of the solution. If we were to match the optimization tasks' characteristics based on the authors, then

- the cost function evaluates the expected outcome of prescribing a treatment at a leaf via an off–policy estimator
- the transition applies the split to create child subproblems
- the comparison maximizes expected utility
- the combination aggregates child utilities (with optional test cost at splits)
- constraints can encode capacity or budget limits

2.4.2 Multi-objective optimization

The ability to optimize multiple objectives in a unified task provides a significant advantage to the STreeD framework. The implementation is explicitly based on a Pareto-front formulation to keep all equally optimal solutions under two or more optimization objectives. However, before we analyze how these kinds of problems are defined by the separability constraints, we cover the preliminaries of the multi-objective optimization and the foundation of Pareto fronts.

Setup of multi-objective optimization

First, to define the multi-optimization problem, without loss of generality, we consider it to be a minimization problem, with an objective mapping $F : X \to \mathbb{R}^m, F(x) = (f_1(x), f_m(x), \ldots, f_m(x))$, over a feasible set X, where $f_m(x)$ is the $m-$th objective, and as a feasibility set we define the set of all possible solutions that satisfy the constraint given by a problem [62].

In such problems, the objectives usually conflict as they handle their minimization separately. For that reason, there is no single solution that minimizes all the objectives at once. Instead, the solution quality is ordered by $\mathbb{R}^m$. The solutions are compared component-wise based on the order of the objective scores. In single objective problems, the order by contrast is characterized as total order, as the only scalar objective is totally comparable for every decision x and the solutions preserve a monotonic order. By definition $y \preceq y'$ is the standard partial order of comparison if for every objective i $y_i \leq y'_i$. Strict order being defined as $y \prec y'$, is used when at least one inequality is strict.

In the same manner, a decision x weakly dominates x' when $F(x) \preceq F(x')$ and strictly dominates it when $F(x) \prec F(x')$. A decision $x^\star$ is considered Pareto-optimal or efficient if there is no other feasible x that dominates it, and the image of all efficient decisions under F is the Pareto-front. Specifically, the strictly efficient and weakly efficient solutions are described by equations 2.4.5 and 2.4.6:

$$X_E := \{x \in X : \nexists x' \in X \text{ with } F(x') \preceq F(x), F(x') \neq F(x)\}, \tag{2.4.5}$$

$$X_{WE} := \{x \in X : \nexists x' \in X \text{ with } F(x') \prec F(x)\}. \tag{2.4.6}$$

Similarly, the minimal Pareto front is defined as $Y_E \subseteq \mathbb{R}^m$, where:

$$Y_E := F(X_E) = \{y \in \mathbb{R}^m \mid \exists x \in X_E : y = F(x)\}, \tag{2.4.7}$$

which is the image of all strictly efficient decisions under the objective mapping. Reporting Y_E, rather than a single scalarized solution, is standard in multi-objective analysis because it summarizes all trade-offs that cannot be improved simultaneously. The advantage of such sets of solutions is that the optimal solutions of the front can be personally chosen to be used as optimal observations, given the importance of one or more objectives over the rest [29]. For example, in figure 2.4.1, if one assumes the *objective* x has a greater importance than *objective* y, then they can concentrate on the solutions closer to the smaller values of the horizontal axis, or if the balance of the trade-off is of greater importance, without the use of scaling the user can concentrate on the sum of the objectives that is represented by points more central on this representation.

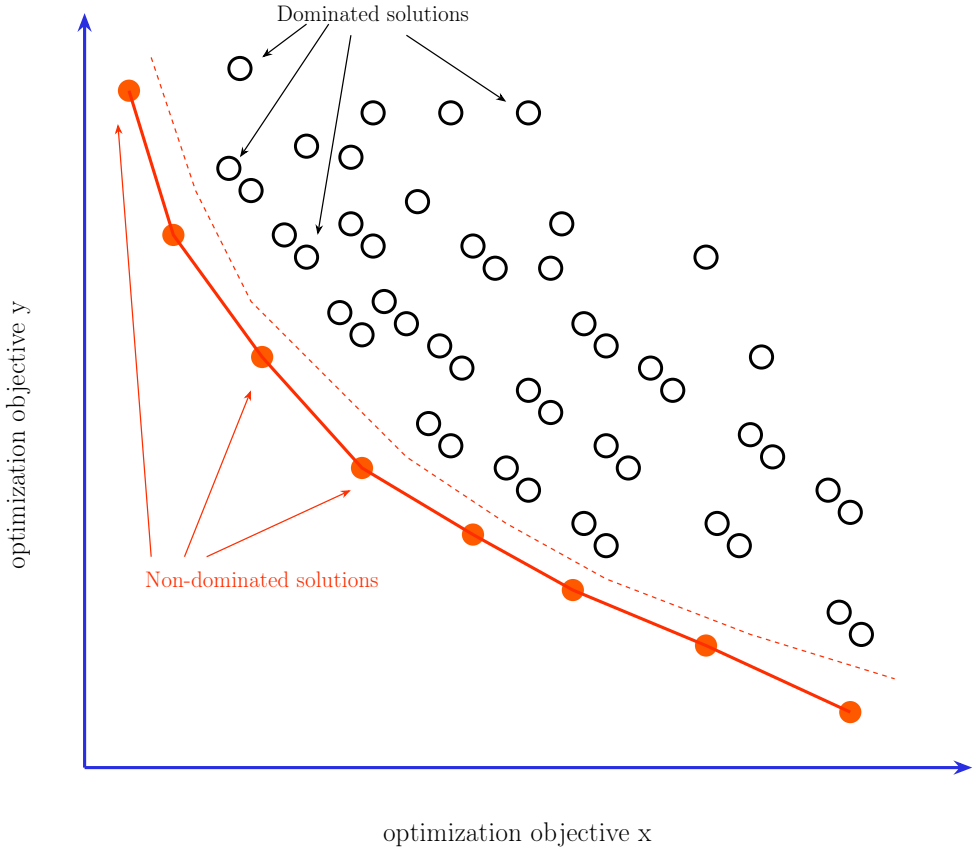

Figure 2.4.1: Pareto front of two minimization objectives

The orange points represent the Pareto front. They are the non-dominated solutions of all possible combinations of the two optimization objectives, while the white hollow ones represent dominated solutions not included in the Pareto front.

At this point, two clarifications of practical importance should be made:

(1) Preference articulation and scalarization are post hoc choices. This formally addresses the previous advantage we referred to, as selecting a single operating point of Y_E can be done after the front is computed, by preferences stated *a posteriori*. The most common device for such a choice, other than empirical preference, is the weighted sum scalarization as depicted by equation 2.4.8

$$\min_{x \in X} \sum_{i=1}^{m} w_i \, f_i(x) \quad \text{with} \quad w_i > 0, \; \sum_i w_i = 1, \tag{2.4.8}$$

which emphasizes the objectives according to weights chosen by the user. However, these sums provably recover only the supported portion of the front. Specifically, given the set of attainable objective vectors $F(X)$ (the image of the feasible set X) and its convex hull $\operatorname{conv} F(X)$, the weighted sum method places a supporting hyperplane to $\operatorname{conv} F(X)$. The optimum lies where the hyperplane and the hull touch, known as an exposed face. Solutions obtained this way are therefore the *supported* efficient points [29]. When $F(X)$ is non-convex (something typical in discrete and combinatorial learning), the Pareto set may contain efficient points lying strictly below the faces of $\operatorname{conv} F(X)$. These are *unsupported* points that no supporting hyperplane can touch. Consequently, weighted sums cannot recover them, and one must resort to alternative scalarizations such as the ε-constraint method or achievement scalarizing functions, or to explicit front enumeration.

(2) Robust alternatives to weighted sums exist to address this loss of information and complement weighted

sums. Two most common techniques are:

$$\begin{aligned} &\varepsilon\text{-constraint:} && \min_{x\in X}\ f_1(x)\ \text{ s.t. }\ f_i(x)\le \varepsilon_i,\quad i=2,\dots,m,\\ &\text{Reference-point:} && \min_{x\in X}\ \max_{i=1,\dots,m}\ \lambda_i\left(f_i(x)-z_i\right), \end{aligned}$$

where $\varepsilon \in \mathbb{R}^m$ sets admissible levels for the other objectives and z is a user-specified aspiration point with scaling $\lambda_i > 0$. Both approaches are standard and allow fine-grained control of trade-offs without assuming convexity[63].

In scenarios, when Y_E becomes large, an ε-dominance filter can be used as a principled approximation: y ε-dominates y' if $y_i \le y'_i - \varepsilon_i$ for all i, which bounds the retained front size at controlled accuracy loss[64].

Multi-objective adaptation in STreeD

In combinatorial decision problems such as decision-tree construction, it is natural to compute and propagate *sets* of non-dominated objective tuples. Each subproblem (sub-tree) returns its local Pareto set. Then the parent nodes form all pairwise combinations of child tuples and then prune dominated outcomes in a process described as "merge, then prune". This paradigm has been used in bi-objective dynamic programming for optimal trees, for example, in fronts over *False Positives, False Negatives*$(\mathrm{FP},\mathrm{FN})$ for nonlinear metrics like the **F1-score** and serves as the direct source to the present framework first presented by [65].

Within this work's framework, we adopt exactly that set-valued view. Given the fixed optimization tuple $o = \langle g,\, t,\, \succ,\, \oplus,\, c,\, s_0\rangle$ (defined previously in section 2.4.1), a state space S, a value space V, features F, and labels K, we assume a current state $s \in S$ and a split $f \in F$. Thus we denote the child states as $s_1 = t(s, f, 1), s_2 = t(s, f, 0)$.

To express the Pareto front relations and operations, the authors defined the following operations. For any candidate sets $\Theta, \Theta_1, \Theta_2 \subseteq V$, the following operators will be used in both the multi-objective and the separability sections:

- **Feasibility filter** $\mathrm{feas}(\Theta, s) := \{\, v \in \Theta \,:\, c(v, s) = 1 \,\}$. This keeps exactly those values that satisfy the task's feasibility predicate at state s (e.g. budget, minimum support, loss thresholds).
- **Non-domination filter** $\mathrm{nondom}(\Theta) := \{\, v \in \Theta \,:\, \nexists v' \in \Theta \text{ with } v' \succ v \,\}$. This removes strictly dominated values and returns the (Pareto) maximal set under
- **Local optimal set** $\mathrm{opt}(\Theta, s) := \mathrm{nondom}\big(\mathrm{feas}(\Theta, s)\big)$. This comprises feasibility and non-domination, and is the set a DP node must keep for combination
- **Merge at a branch** $\mathrm{merge}(\Theta_1, \Theta_2, s, f) := \{\, v_1 \oplus v_2 \oplus g(s, f) \mid v_1 \in \Theta_1,\ v_2 \in \Theta_2 \,\}$. This forms all parent candidates from one value of each child, adding any branch-level contribution.

Given these operators, the authors define the set-valued recursion as:

$$T(s,d) = \begin{cases} \mathrm{opt}\big(\{\, g(s,\hat{k}) : \hat{k} \in K \,\},\ s\big), & d = 0,\\ \mathrm{opt}\Big(\bigcup\limits_{f\in F} \mathrm{merge}\big(T(t(s,f), d-1),\ T(t(s,\bar{f}), d-1),\ s,\ f\big),\ s\Big), & d > 0. \end{cases} \tag{2.4.9}$$

Equation 2.4.9 states that every subtree of depth d, rooted at state s, is summarized not by a single score but by the full set of feasible, non-dominated objective tuples that this subtree can realize. At depth $d = 0$ (a leaf), the only decisions are label assignments $\hat{k} \in K$, so the node returns the opt set over those assignments. For $d > 0$, an internal node considers each possible split $f \in F$, takes the Pareto sets returned by its two children $T(t(s, f), d-1)$ and $T(t(s, \bar{f}), d-1)$, combines them through merge, and then applies opt to discard any infeasible or dominated combinations. In other words, the parent does a "merge, then prune" step over all its children's possible trade-offs.

A direct consequence is that the root state s_0, evaluated at the target depth budget $d_{\max}$, returns the entire (multi-objective) Pareto front that can be obtained by any decision tree of depth at most $d_{\max}$. This is strictly stronger than returning a single tuned model: it exposes every non-dominated trade-off between objectives

such as accuracy vs. cost, false positives vs. false negatives, or predictive performance vs. fairness, without committing in advance to one particular weighting of those objectives.

It is also important to note that the size of $T(s, d)$ (could be described as the front width) can, in principle, grow combinatorially, because we keep *all* undominated objective tuples. This is the standard price of computing exact Pareto fronts in discrete problems, and it motivates pruning techniques such as caching, bounding, or (when allowed) approximate dominance like ε-dominance to limit explosion in practice.

In the next subsection section, we will complete this related work brief analysis via one of the most crucial references, that of separable problems. This last piece of information is crucial for our later method proposal and formulation. To define our task, under the framework of the STreeD solver, the separability constraints and properties were closely studied and later proven to match our problem setting and needs.

2.4.3 Separability

We now formalize the constraint under which set-valued dynamic programming over trees is *both* correct (returns a globally optimal set at each node) and efficient (permits caching/pruning without cross-sibling couplings). The framework refers to such tasks as *separable*.

We begin by recalling the optimization task tuple o and the two child states s_1, s_2 as:

$$o = \langle g,\ t,\ \succ,\ \oplus,\ c,\ s_0 \rangle,$$

$$s_1 = t(s, f, 1), \qquad s_2 = t(s, f, 0),$$

with child states directly deriving from a parent state $s \in S$ and split $f \in F$. We also consider the same value space, feature space, and label set, respectively, are V, F, K.

We will use the operators $\text{feas}, \text{nondom}, \text{opt}, \text{merge}$ from Section 2.4.2.

Definition (Task Separability). A task o is *separable* iff, for every $s \in S$, $f \in F$, and candidate sets $\Theta_1, \Theta_2 \subseteq V$ representing child solutions at s_1, s_2,

$$\boxed{\text{opt}\left(\text{merge}(\Theta_1, \Theta_2, s, f),\ s\right) \ =\ \text{opt}\Big(\text{merge}\left(\text{opt}(\Theta_1, s_1),\ \text{opt}(\Theta_2, s_2),\ s,\ f\right),\ s\Big).} \tag{2.4.10}$$

By this, we describe that the parents' optimal set can be obtained by *first* optimizing each child independently at its own state and *then* combining, rather than optimizing only after a Cartesian combination of *all* child candidates. STreeD proves that the following three properties are *necessary and sufficient* for equation 2.4.10 to hold:

1. **Markovian local evaluation and transition.** The node evaluation and transition depend only on the current state and the immediate action/branch:

   $$g(s, a) \text{ depends only on } (s, a), \qquad t(s, f, \delta) \text{ depends only on } (s, f, \delta),$$

   with no dependence on off-path or sibling decisions. This ensures well-defined subproblems and allows memoization: identical child states yield identical optimal sets, regardless of how they were reached.

2. **Order preservation of the combination operator $\oplus$.** Improving a child while holding the other fixed must not worsen the parent:

   $$v_1 \succ v_1' \ \Rightarrow\ v_1 \oplus v_2 \succ v_1' \oplus v_2 \quad \text{(and symmetrically if } \oplus \text{ is non-commutative)}.$$

   Additivity under minimization is a special case (with $+$ and component-wise order), but the requirement also admits non-additive/partially ordered objectives so long as $\oplus$ cannot invert dominance.

3. **Anti-monotonic feasibility.** If a child's value is infeasible at its child state, then no parent combination containing it can survive:

   $$\neg c(v_1, s_1) \text{ or } \neg c(v_2, s_2) \implies v_1 \oplus v_2 \notin \text{opt}(\Theta, s), \text{ with } \Theta = \text{merge}(\Theta_1, \Theta_2, s, f).$$

   This justifies local feasibility filtering (e.g., minimum support, budget thresholds) without risking the removal of building blocks of any globally optimal solution.

Based on the properties provided by the separable problem definition, we need to address some additional context of multi-objective problems, given the STreeD setting. Closure under a combination of tasks is important because, in practice, we rarely care about a single objective. We may want accuracy and fairness at the same time, or we may want treatment quality and treatment cost. The framework needs to allow us to optimize such multi-objective problems in one tree without breaking the guarantees we have just established. Formally, assume we have two tasks $o^{(1)} = \langle g^{(1)}, t, \succ^{(1)}, \oplus^{(1)}, c^{(1)}, s_0 \rangle$ and $o^{(2)} = \langle g^{(2)}, t, \succ^{(2)}, \oplus^{(2)}, c^{(2)}, s_0 \rangle$ that are both already known to be separable on the same underlying state and transition structure. We then construct a joint combination task, which we denote $o^{\otimes}$, whose objective space is simply the concatenation of the objectives of $o^{(1)}$ and $o^{(2)}$. Concretely, we define the task components as:

- $g^{\otimes} = (g^{(1)}, g^{(2)})$,
- $\succ^{\otimes}$ as the product (component-wise) order,
- $\oplus^{\otimes}(v, w) = \big(v \oplus^{(1)},\, w \oplus^{(2)}\big)$,
- $c^{\otimes} = c^{(1)} \wedge c^{(2)}$.

The point is that if the tasks are individually separable, then their combination can also be separable. This holds as long as the combining operator maintains order under $\succ^{\otimes}$ and each constraint $c^{(i)}$ is anti-monotonic as described before. In simple terms, nothing changes when we combine two separable tasks. If feasibility fails in either child task, it still fails for the parent. Dominance in the combined space is simply dominance for each component, and better solutions for the child tasks do not make the parent worse. Therefore, optimizing multiple objectives concurrently is like optimizing a higher-dimensional version of the same dynamic program. The construction of the Pareto set over $(m_1 + m_2)$ objectives is still valid based on the same recursion.

This shows that separability is not just a technical detail, but a fundamental requirement for this approach to work. When the properties of separability hold, the parent node of the tree can be calculated through a straightforward split-and-merge process. Each child sub-tree is solved only once from its own state, and its opt set is maintained. These sets are then merged using $\oplus$ along with the local branching contribution $g(s, f)$. Finally, the parent applies feasibility and non-domination to eliminate anything that is either infeasible or dominated. This method is both correct and efficient. It is correct because the parent receives every globally optimal value it could achieve, and efficient because we can store subproblems, eliminate dominated partial solutions early, and avoid any unnecessary coordination between siblings.

2.5 Interpretable Trees

In this section, we delve into a necessary brief analysis of the interpretable aspect that the tree algorithms offer. To determine the interpretability of an ML model, we refer to it as the degree to which humans can understand the cause of a decision by the algorithm, or predict its output given an input as presented by [66–68]. In this setting, a model is interpretable when its internal reasoning can be followed and verified without any additional post-hoc mechanisms, meaning that there is no need for additional methods that provide explanations to outputs after the decision of the model has been returned.

An internal node in a decision tree evaluates a feature condition such as $x_j \leq \tau$. This makes it clear to the reader that the instance space is being partitioned at a specific threshold of the feature x_j. By arranging such tests hierarchically, the model forms root-to-leaf paths that act as concrete rules mapping observable constraints to the outcome written at the leaf. A simple illustration helps. If income $>$ 30k and debt ratio ≤ 0.3 then accept. The value of a tree lies in this traceability. A prediction is justified by the exact path the input follows, and the entire rule set can be inspected, drawn, and audited step by step.

At this point, a thing that remains undefined is the practical need to quantify interpretability so that models can be compared in a principled way. A common approach is to use structural complexity as a proxy. Depth and size determine how many distinct paths exist, and a larger collection of paths makes complete human cognitive simulation harder for a reader. For this reason, many works use simple surrogates based on the number of nodes or leaves, as if simplicity or complexity derives from $\frac{1}{|T|}$, where $|T|$ is the cardinality of the chosen structural unit, that is, the internal or leaf nodes. This can be read as an informal indicator rather than a formal law, entirely for the sake of intuitive understanding. In practice, small and shallow trees are

easier to follow. Deeper structures often trade some transparency for additional predictive flexibility, often guiding the decision of the model to overfit on training samples. The aim is not to minimize size at all costs but to keep the model within a range that allows the user to have an end-to-end track of the structure.

Additionally, interpretability is not considered a single attribute. It has several aspects that matter in practice. One can ask whether a reader can follow a complete decision for a single case from start to finish, or whether the parts that make up the model, such as features or the thresholds, are individually understandable. A further question is whether the learning procedure itself is transparent, so that the way the model is produced is clear. These views, often described as simulatability, decomposability, and algorithmic transparency, give a useful lens for decision trees and will guide the design choices that follow [66, 69].

The way a tree splits the feature space has a direct effect on readability. Classical univariate trees use a single variable at each internal node, for example $age \leq 45$. This keeps each test easy to read and preserves a simple connection between data and decision [3]. Oblique trees use linear combinations of features, written as $\mathbf{w}_t^\top \mathbf{x} \leq \theta_t$. Such tests can shorten paths and reduce depth, which may help at the global level, yet they make a single node harder to interpret because several features participate at once [70]. A single multivariate node can therefore reduce the decomposability of every rule that passes through it [66]. To soften this trade-off, restrictions on the number of active coefficients or the use of additive forms so that each test involves only a few features were introduced by [71, 72]. Also, another option is to phrase conditions with graded membership rather than hard thresholds, which yields fuzzy rules that attach a degree of truth in $[0, 1]$ to statements like $x_j \leq \theta_t$ and allows partial membership of an instance in more than one branch [73, 74].

However, the interpretability of a node's thresholds and rules alone, leave gaps. The way inputs are routed also matters. In the usual hard setting, each example follows one path to a leaf, and there is a one-to-one link between the evidence and the outcome. In soft or probabilistic trees the router at a node is a smooth function of the features, for instance

$$P_L(\mathbf{x}) = \sigma(\mathbf{a}_t^\top \mathbf{x} - b_t), \qquad P_R(\mathbf{x}) = 1 - P_L(\mathbf{x}), \qquad \sigma(u) = \frac{1}{1 + e^{-u}}.$$

This makes predictions locally smooth with respect to the input and reduces discontinuities around split boundaries. The price here is a drop in simulatability because there is no single deterministic path that explains the output for a given case. Fuzzy decision trees use the same idea to represent uncertainty explicitly and to trade crisp rules for continuity when data are noisy [75, 76].

As previously stated, there is a growing body of work that trains tree structures with gradients. These models learn the parameters of the split functions, and sometimes the structure itself, using backpropagation. The goal is to keep the familiar skeleton of a tree while gaining some of the predictive strength that comes with continuous optimization [77, 78]. Such training blurs the sharp boundaries that make classical trees easy to read. This way, one can recover part of the interpretation by projecting learned hyperplanes to a small set of dominant features or by assigning semantic concepts to internal nodes so that each branch corresponds to a recognizable notion [79–81].

Finally, stability is another practical dimension to determine the ease of understanding of the tree structure. Small changes in the training sample can lead to different trees, especially when there are many nearly tied splits. This makes auditing harder and diminishes user trust, since similar inputs may appear to be governed by different rules [82]. Simplest approaches to handle such implications include regularization that discourages near ties, splitting strategies that group similar cuts into segments, and formulations that encourage consistent structures across runs [83, 84]. The goal is not absolute invariance but a model that yields rules stable enough to be checked and discussed.

Interpretability can also extend beyond structural clarity to questions of cause, reliability, and action. Causal decision trees aim to uncover heterogeneous effects under potential interventions and support counterfactual reasoning about what would change under a different action [85]. Conformal prediction trees attach calibrated uncertainty to leaves so that each prediction is accompanied by an interval or set that has a specified coverage level [86]. Counterfactual decision trees report minimal changes to inputs that would flip the prediction, which links a rule to concrete steps a user might take [87].

As far as we are concerned, the latter approach of counterfactual explanation trees, is the field that this thesis was mainly influenced by, and the following work derives from implementations that will be further discussed

in Section 3.4, where we will present the framework of Counterfactual Explanation Trees (CET), the way it operates and the advantages it provides to user interpretability of model decisions. Nevertheless, before we analyze this, we will first set the ground for the broader domain of Explainable Artificial Intelligence and its preliminaries in the initial sections of Chapter 3.

Chapter 3

Explainable Artificial Intelligence

Explainable artificial intelligence (XAI) tackles a basic tension in modern machine learning. High-performing models often behave like black boxes, which makes validation, contestation, and oversight difficult in sensitive applications. Legal and institutional expectations about accountability and transparency, together with the ongoing discussion of a GDPR right to explanation, increase the demand for methods that make automated decisions understandable to the people they affect [88]. Beyond regulatory aspects, explanations support error analysis, robustness checks, which contribute to the calibration of trust, especially when failures to prediction carry real costs that affect human lives, for example, a healthcare application that contributes to the remedy of a patient or a legal application that might provide verdicts through racial bias [68, 89]. In such settings there is a strong case for choosing models that are interpretable by design, rather than relying on explanations constructed after the fact [90].

XAI as a domain is a spectrum, where one end has models that are transparent by design, where the logic is visible. At the other end lie post hoc explanations that aim to provide faithful and understandable accounts of otherwise opaque predictors [68, 89]. Explanations focus on a single instance, providing a local view of users to understand what influence neighboring instances have and how decision boundaries influence individuals. On the other hand, they can summarize behavior across populations or partitions of them, which provides a global view. Sufficiency turns on two criteria that often pull in different directions, faithfulness to the underlying predictor and intelligibility for the intended audience and task, so a deliberate balance is required.

When considered based on methodology, XAI gathers several groups of methods, each with distinctive assumptions. Feature attribution methods assign scores that measure local importance or build additive decompositions, with fidelity shaped by the treatment of feature dependencies and the choice of estimator [91]. Example-based and prototype-based approaches rely on case-based reasoning, concept-based analyses probe internal representations using human-aligned concepts, and model simplification replaces complex predictors with more readable surrogates, such as rule sets and trees, trading some global fidelity for comprehensibility [89, 92]. Each category exposes distinct failure modes, including instability, sensitivity to distribution shift, and semantic mismatch, which motivates careful evaluation and clearly stated validity regimes [91].

Counterfactual explanations occupy a distinctive place among contrastive approaches because they adopt a prescriptive and action-oriented view. They describe the smallest changes to features that would flip an outcome, which links explanations to concrete recourse and strengthens institutional accountability [92]. This angle ties explainability to intervention and feasibility, so attention shifts to data realism, fairness at the individual and group level, and the operational cost of recommended changes [25].

Actionable recourse builds on this foundation, and recourse *summaries* gather many individualized prescriptions into compact and transparent artifacts, for example, rule lists or trees that can be audited and deployed [28]. The rest of the chapter follows this thread in order. Core definitions and methods come first. Formal treatments of counterfactual explanations and actionable recourse follow. The chapter closes with globally comprehensible rule-based summaries that aim to reconcile interpretability with the demands of

decision support [90].

Contents

3.1 Background, Definitions, Approaches

Modern AI systems and, in particular, deep architectures and large ensembles, have achieved state-of-the-art predictive and generative performance at the expense of having high-performance black-box frameworks. This unclear view of the algorithmic internal decisions of models has motivated an extensive amount of work on XAI, aiming to make the model behavior understandable to humans for tasks such as validation, debugging, trust calibration, and governance [66, 89, 93]. As already stated, beyond scientific curiosity, the need for explanations is driven by societal and regulatory demands for accountability and transparency in high-risk decisions, such as those in healthcare, finance, and justice. The right to explanation has become more demanding and more essential than ever for every person, directly or indirectly affected by the actions of automated decision systems.

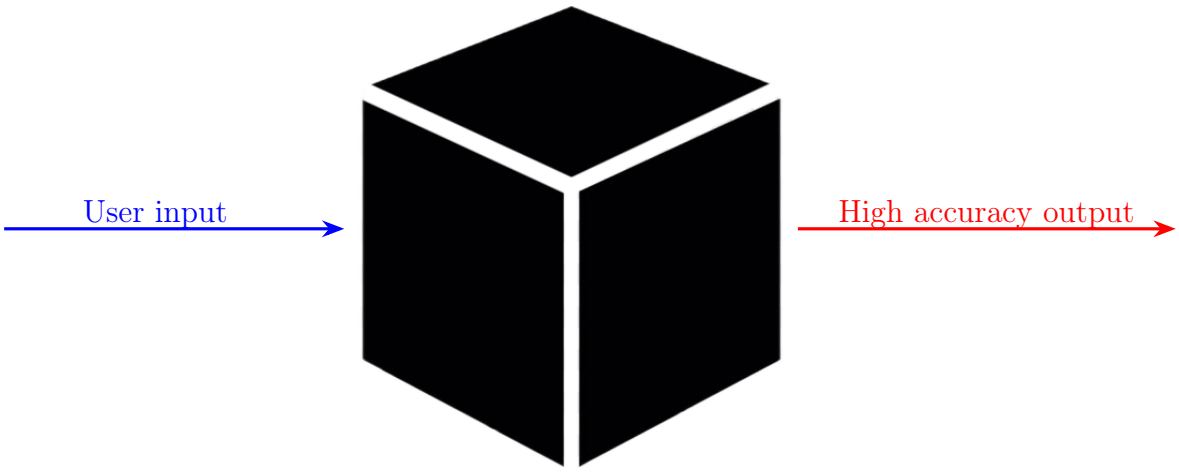

Figure 3.1.1: A model's behaviour illustrated as a black box

In this section, we provide the necessary definitions and setting of the field of XAI. We then briefly cover the domains and approaches of XAI, providing the respective context on the impact they have had over the years and the continuous, still-growing need for explainability. At the end of the section, we present the methodological approaches that have had significant importance until this day, and continue to be entrusted tools that provide transparency to users, helping people, not only evaluate the performance of a model on bias or maltreated training data, but also in making crucial actions that the average human would not have considered directly making.

3.1.1 Definitions and Foundations

Let $f : \mathcal{X} \to \mathcal{Y}$ denote a trained predictor and $\mathcal{D}$ a data distribution. We follow established taxonomies [66, 93] and make the following working distinctions.

- *Transparency* aggregates properties that render the internal mechanism of f directly understandable. A standard decomposition distinguishes (i) *simulatability* (a human can mentally execute f on typical inputs), (ii) *decomposability* (parameters, features, and intermediate computations have explicit meaning), and (iii) *algorithmic transparency* (training and optimization behave predictably) [66]. These facets are architectural: they concern f itself rather than a separate explanatory object.
- *Interpretability* is the degree to which a human can predict or comprehend f's behavior. It is often operationalized via cognitive-load constraints such as sparsity, small depth, or short rule length, reflecting the need for simulatability under bounded attention [90]. Interpretability is audience- and task-dependent.
- *Explainability* refers to mechanisms that produce human-comprehensible reasons for f's outputs. Formally, an explanation method $\mathcal{E}$ maps an instance–model pair to an explanatory object $e = \mathcal{E}(x, f)$ intended to be (i) *faithful* to f and (ii) *intelligible* to the target audience [68, 93].

Two scope regimes are customary. *Local* explanations target behavior in a neighborhood Π_x of a query x, often by fitting a simple surrogate g_x with small local discrepancy $\mathbb{E}_{z\sim\Pi_x}[\ell(f(z), g_x(z))]$. *Global* explanations summarize behavior over $\mathcal{D}$ via a single interpretable model g with small $\mathbb{E}_{z\sim\mathcal{D}}[\ell(f(z), g(z))]$ or a structured synopsis of decision logic [16, 94]. Intrinsically interpretable models provide global transparency by design; post-hoc methods typically offer local views that can be aggregated under additional regularity assumptions [89].

Beyond scope, three evaluation axes recur:

1. *Faithfulness* (F). For a decoder ϕ that renders explanations executable (e.g., a rule evaluator),

$$\mathsf{F}(x; \mathcal{E}, f, \Pi_x) \;=\; 1 - \mathbb{E}_{z\sim\Pi_x}\big[\,\ell\big(f(z),\, \phi(\mathcal{E}(x, f), z)\big)\big]\,,$$

   where ℓ measures predictive discrepancy. High F indicates that the explanation reliably reproduces f's local or global behavior [93].

2. *Stability/Robustness* (S). Small perturbations $x' \approx x$ should not produce qualitatively different $e' = \mathcal{E}(x', f)$. Stability is commonly probed via $d(e, e')$ under controlled input shifts, guarding against explanations that are overly sensitive or manipulable [91, 92].

3. *Intelligibility/Complexity* (I). Cognitive accessibility is often proxied by description length $L(e)$ (e.g., number of rules, nonzeros, or tree depth). Explanations should keep $L(e)$ within human limits without sacrificing F [90].

These criteria are interdependent: simplifying e to improve I can degrade F; enforcing stability can constrain the admissible form of $\mathcal{E}$. The balance is application-specific and hinges on audience, stakes, and downstream use (e.g., audit vs. individual contestation) [68]. It is also essential to distinguish *associational* explanations from *causal* accounts: many XAI methods describe correlations in f's decision boundary, whereas causal explanations address *interventions* and counterfactuals in a structural model of the data-generating process [20].

3.1.2 Domains and Approaches of XAI

XAI is deployed across application classes with distinct risks, stakeholders, and desiderata:

- *Classification and decision support*: reliable rationales, calibrated uncertainty, and traceable failure modes; in high-stakes settings, the emphasis is on verifiable faithfulness and conservative deployment.
- *Recommendation and personalization*: transparency to reduce manipulation, expose preference drivers, and support user recourse (e.g., how to alter recommendations).
- *Healthcare*: explanations for accountability, safety, and alignment with clinical knowledge; justifications must be compatible with care pathways.
- *Finance and credit scoring*: explanations underpin adverse action notices, enable fairness auditing, and support compliance.
- *Justice systems*: explanations for contestability and due process, enabling scrutiny for bias and subgroup calibration.
- *Autonomous and safety-critical systems*: interpretable rationales for assurance cases, post-incident analysis, and human–machine teaming; emphasis on stability and robustness.
- *Scientific discovery and data-centric research*: explanatory artifacts (e.g., sparse rules, salient features) can generate testable hypotheses but must be distinguished from causal claims.

These contexts motivate different emphases: robustness to distribution shift and adversarial perturbations, actionable guidance and user recourse, causal plausibility, privacy preservation, and group/individual fairness constraints. Selecting and evaluating XAI methods therefore requires making explicit the target scope (local vs. global), the fidelity–intelligibility trade-off, and the normative assumptions (associational vs. causal) appropriate to the domain [68, 88–90].

3.1.3 Methodological Approaches

It is useful to state, for each method family, its objective and the assumptions under which its guarantees hold [21, 91, 92].

Feature attribution

Feature attribution estimates the contribution of each input coordinate to a specific prediction. A common approach is to approximate a complex predictor f in a small neighborhood of a query x and read off per-feature effects. LIME does this by fitting a sparse local surrogate g_x with an exponential locality kernel $\pi_x(z) = \exp(-\|z - x\|^2/\sigma^2)$ and an ℓ_1 penalty [16]. Shapley-style methods average marginal contributions over coalitions to satisfy axioms, conditional estimators address dependent features [17, 95, 96].

Reliability hinges on design choices. The locality kernel and bandwidth σ define what counts as "local" and can induce instability. In Shapley estimators, the background distribution $p(z)$ determines how dependence is handled and can shift attributions even when f is fixed [95]. Additive decompositions compress interactions; interaction-aware variants (e.g., Shapley–Taylor) make such effects explicit [97]. Sensitivity checks, agreement under perturbations, and variance reporting are recommended, and global influence tools provide a complementary view of input effects [98].

Prototype and example-based explanations

When concrete cases are preferred over abstract scores, prototype–criticism frameworks select representative instances that support a prediction and contrasting cases that argue against it [67]. Coverage matters: selection based solely on geometric proximity or density may miss the decision boundary of f. Performance improves when selection is guided by margins or influence, diversity constraints prevent near duplicates, and diagnostics flag under-covered regions [91, 99].

Concept-level and representation explanations

Concept-based methods elicit human-aligned concepts and probe the sensitivity of f along corresponding directions in representation space, linking semantics to internal features [92]. Sound use requires (i) well-specified, reliably instantiated concepts, (ii) directions that capture meaningful variation, and (iii) on-manifold edits so changes remain realistic. Control concepts, ablations, and counterfactual data augmentation help detect shortcuts and spurious sensitivity [21]. Some notable works made to achieve concept-level explanations are [100–102] and representation explanations are[102–105]

Model simplification and surrogates

To summarize overall behavior, global surrogates approximate f with interpretable models (e.g., small trees or sparse rule lists) and report where the surrogate is trustworthy. High-precision local rules can be composed into readable summaries with coverage and precision guarantees [106]. Two cautions apply: fidelity and simplicity trade off, and summaries must state their domain of applicability. Reporting coverage with calibrated fidelity avoids an illusion of universality and clarifies reliability across regions [107].

Contrastive and counterfactual explanations

For the question "what small change flips the outcome?", solve

$$\min_{\delta \in \mathcal{F}} c(\delta) \quad \text{s.t.} \quad f(x + \delta) = y^*.$$

Here $c(\delta)$ encodes proximity or sparsity, and $\mathcal{F}$ encodes feasibility in the real world [91]. Once framed as recourse, design choices are normative: immutable attributes should be protected, and monotonic relations should reflect domain logic. Institutional rules and budgets affect realism and fairness in who can act. Distribution-aware objectives keep proposals on-manifold and reduce brittle edits; optimization-based generators can enforce density or support constraints when realism is critical [24, 25].

3.1.4 Methodological Approaches Across ML Subfields

We relate the above families to core ML subfields, stating typical use, the rationale for fit, and concrete validity checks, with focused citations.

Supervised learning on tabular data

Attribution, global surrogates, and counterfactual/recourse methods are natural here. Features often correspond to editable attributes and constraints follow domain policy. In practice, report surrogate *coverage* and *fidelity* together, and for recourse report flip rate and average cost under a feasibility set encoding immutability, monotonicity, and a budget B [25]. To control brittleness, reject actions with Mahalanobis distance to the training manifold $> 3\sigma$. This thesis focuses on counterfactual explanations for tabular data affecting individual recourse; Section 3.3 introduces the related work, and Section 3.4 provides a detailed comparison.

Computer vision

Saliency can appear plausible while being model-agnostic; therefore, randomization-based sanity tests are a prerequisite—maps should degrade under weight/data randomization [108]. Faithfulness is then quantified with deletion/insertion curves (area-under-curve) via region perturbations [109]. Concept activation analyses and prototype illustrations complement pixel attributions when localization is insufficient.

Natural language processing

Token/span attributions and example rationales support contestation; concept/template probes add syntactic or discourse structure. Subword tokenization and long-range dependencies complicate faithfulness, so stability analyses under paraphrase/noise and human-grounded tasks (e.g., comprehensiveness/sufficiency tests) should be reported [91]. Short diagnostics help: one counter-example often reveals spurious cues.

Recommender systems and personalization

Local rules, prototypes, and counterfactual-style advice align with user goals to understand and change outcomes. Explanations must reflect exposure and feedback dynamics; otherwise they misattribute platform effects. Recommended practice: evaluate with user studies or controlled offline counterfactual protocols, and report whether advice changes behavior beyond exposure baselines [18].

Sequential decision making and reinforcement learning

Explanations target policies and trajectories, not single predictions. State/action attributions should account for transition dynamics and discounting; prototype trajectories and what-if rollouts are informative only with calibrated environment models. Counterfactual reasoning over alternative actions/histories makes the dependence on dynamics estimation explicit and should be paired with uncertainty reporting [110].

Generative modeling

Concept-level probes and latent edits are primary tools. Diagnostic use identifies encoded concepts; prescriptive use steers outputs along validated directions. To avoid off-manifold artifacts, enforce reconstruction constraints and report edit success (target classifier agreement) alongside realism checks (e.g., FID shift under edit magnitude) [92].

Counterfactual explanations thus complement descriptive XAI by specifying feasible input changes that alter outcomes. The next section develops this prescriptive turn in detail.

3.2 Counterfactual Explanations

Counterfactual Explanations have a crucial position in XAI, as they reform model behavior in terms of stating the counterfactual question "what-if". These scenarios are applied on instances x, as we seek a minimal and feasible change to flip its predicted outcome to a desired label. Its foundations derive from formal counterfactuals and interventions, whereas the ML adaptation is represented as a constrained optimization

that delves between properties like proximity, sparsity, or feasibility of instances, given an instance space and the feature distributions. However, counterfactuals are distinguished between two separate fields. The first one is that of the algorithmic approach, which is defined relative to a fixed predictor under observational data, while the second one is based on causal interventions, that is defined within a structural causal model with explicit interventions and identification assumptions [20, 91].

In the following section, we desire to address the necessary preliminaries that counterfactual explanations carry. Provide the formal definitions for both approaches and address the differences between algorithmic and causal counterfactuals. Then we distinguish the four properties that characterize the quality of such explanations, which are their proximity, sparsity, plausibility, and feasibility. Finally, we specify the paradigms on which counterfactual explanations apply to optimization families that operationalize these objectives, and the evaluation criteria used to compare them across tasks.

3.2.1 Origin and Definitions

The origin of counterfactual explanations is beyond machine learning. Specifically, posing the "what-if" statement question derives from two long-standing fields of theoretical background. The most common origin stands on the field of cognitive psychology, where counterfactual thinking means mentally simulating alternatives to past events. On the other hand, in philosophy and logic, the counterfactual conditionals offer formal semantics for specific reasoning, that of the "nearest-world"[111, 112]. This broad lineage is the general motivation, behind the definition and formulation on the approaches of algorithmic and causal fields of explanations of models.

Causal definition

The present the causal definition, we first need to consider the Structural Causal Model (SCM) formulation. As formulated by [20], an SCM is defined as $\mathcal{M} = \langle U, V, F, P(U) \rangle$. The arguments of SCM M, are categoried as exogenous variables U, endogenous variables $V = \{V_1, \ldots, V_n$, structural assignments $F = \{f_1, \ldots, f_n$ interpreted as $V_i = f_i(pa_i, U_i)$ with $pa_i \subseteq V \setminus V_i$, and a distribution $P(U)$ over exogenous variables.

For $A \subseteq V$ and an intervention do$(A = a$, the unit-level counterfactual is $Y_{A \leftarrow a}(u)$, meaning that the value Y is obtained by solving the intervened system at exogenous context u. We define the admissible action space $\mathcal{A}(x)$ to encode which variables in A can be set and how, depending on bounds, immutability, etc. Also, we enforce causal feasibility through $\mathcal{G}$ as no direct manipulation of non-accessible descendants and all downstream changes must arise via the structural equations F. By this we obtain a causal counterfactual explanation as

$$\min_{a \in \mathcal{A}(x)} \tilde{C}(a) \quad \text{s.t.} \quad Y_{A \leftarrow a}(u) = y^\star, \quad a \text{ respects } \mathcal{G} \text{ (graph constraints)}, \tag{3.2.1}$$

where $\bar{C}(a)$ quantifies intervention effort.

Algorithmic definition

Given a trained predictor $f\, X \to Y$, a target label y^*, an instance $x \in X$, and a feasibility set $\mathcal{F}(x) \subseteq X$, that constraints the process by encoding the domain, the mutability of features, the monotonicity, and budget constraints, we define as (nearest) counterfactual explanation as any solution of:

$$\min_{x' \in \mathcal{F}(x)} C(x, x') \quad \text{s.t.} \quad f(x') = y^*. \tag{3.2.2}$$

Here C is a cost specified by the user, such as a weighted norm (like ℓ_1, ℓ_2 norms), sparsity penalties, or mixed-integer edit costs.

Another formal and complete approach by [26] is

$$\min_{x'} \underbrace{\mathcal{L}\big(f(x'), y^\star\big)}_{\text{target loss}} + \lambda \underbrace{d(x, x')}_{\text{proximity}} + \Omega(x') \quad \text{s.t. } x' \in \mathcal{F}(x), \tag{3.2.3}$$

where $\mathcal{L}$ is a margin-based loss and Ω represents the optional regularizers for plausibility.

The algorithmic approach is tightly accompanied by four properties. A satisfactory counterfactual set should meet:*Proximity, Sparsity, plausibility, and feasibility*, each constraining a different aspect of the quality of such explanations.

In more detail, *proximity* requires the edited instance x to remain close to the original query, which is formalized as a weighted metric similar to the distance between the two instances, as $d(x, x')$, so that d is as small as possible. The use of the proximity metric varies and depends on the user's personal need for their task, however, the choice of any metric should be justified in order to explain what "near" means to any given original instance. Secondly, *sparsity* demands that only a few coordinates of the feature vector change (in applications denoted as vector indices), which is typically encoded as $||x' - x||_0$ being small, which should be enforced in data context as its only purpose is to ensure interpretability and return of solutions in manifold.

Plausibility, on the other hand, quantifies how realistic an edit is, by requiring that the new instance lies in a high-density area of the data or a learned manifold. Unlike the proximity property, plausibility must be enforced explicitly via constraints or parameterizations, as it does not follow automatically as a distance metric.

Feasibility, ensures that proposed counterfactuals respect domain and policy constraints, such as immutability, or directional relations, by restricting a feasibility set $x' \in \mathcal{F}(x)$. Such feasibility constraints are the coordinates of a vector that determines the gender identity of an individual, or their race, as far as immutability is concerned, and on the other hand a directional proposal on the education level of an individual should only propose a strictly monotonic increasing edit. These properties interact, as proximity and sparsity together improve cognitive economy, and plausibility and feasibility provide a guard against invalid or unethical prescriptions. However, such interactions lead to trade-offs that should be surfaced rather than hidden behind hyperparameter choices that have been stated to work out [91].

To intuitively comprehend the issue, let a linear classifier be $f(x) = \mathbb{1}[w^\intercal x + b \geq 0]$ with $\mathcal{F} = X$, and the Euclidean cost $c(x, x') = ||x - x'||_2$. Then the nearest counterfactual for a negative label x is the orthogonal projection onto the decision hyperplane:

$$x^\star = x - \frac{w^\top x + b}{\|w\|_2^2}\, w, \qquad f(x^\star) = 1, \tag{3.2.4}$$

which generalizes to weighted norms and box constraints via standard Karush-Kuhn-Tucker (KKT) conditions that are a set of necessary conditions for a solution to be optimal on a local optimization problem, with linear constraints.

Two plain caution observations help avoid common mistakes. First, algorithmic counterfactuals inform the user what the features would need to look like to flip the model's prediction, but they do not tell the user that these changes are realistically achievable by individuals. By contrast, causal counterfactuals specify *actual interventions on controllable causes* that respect a causal graph $\mathcal{G}$ [20]. Mixing these up leads to infeasible advice, for example a proposed edit leads one to change their age. Second, sparsity and plausibility pull in opposite directions. Forcing as few edits as possible ($\|x' - x\|_0$ very small) can produce unrealistic points off the data manifold, while enforcing realism may require allowing a few more coordinated changes. In practice, one should state which objective they prioritize and how they enforce the other, for example, density thresholds or generative constraints for plausibility.

3.2.2 Difference between causal and algorithmic approach

Algorithmic counterfactuals and causal counterfactuals answer related but distinct questions. The algorithmic view asks what alternative input $\tilde{x}$ would make a trained predictor satisfy $f(\tilde{x}) = y^\star$ for the case at hand. The causal view asks what intervention on an admissible set of variables would make the real outcome Y attain $y^\star$ for this unit, which requires an explicit model of the data generating process and the semantics of interventions [20].

A compact way to expose the gap is to compare feasible sets. The algorithmic formulation typically permits edits from a user defined set $\mathcal{F}(x) \subseteq \mathcal{X}$. The causal formulation first specifies a structural causal model $\mathcal{M} = \langle U, V, F, P(U) \rangle$ with a graph $\mathcal{G}$, then restricts actions to admissible interventions $a \in \mathcal{A}(x)$ on actionable

ancestors of Y and finally maps interventions to realizable inputs through the structural equations. Writing $\phi(a; x, \mathcal{M})$ for this mapping, a principled feasible set becomes

$$\mathcal{F}(x) = \{\phi(a; x, \mathcal{M}) \,:\, a \in \mathcal{A}(x)\}.$$

Without such a construction, an algorithmic counterfactual may propose edits to non-actionable descendants or violate monotonicities encoded by $\mathcal{G}$ [25].

Causal counterfactuals also carry identification requirements. Unit level statements $Y_a(u) = y^\star$ depend on exogenous context u or on assumptions that permit identification of interventional quantities from observational data. In the absence of credible structure and identification, algorithmic counterfactuals should be presented as decision boundary guidance rather than guarantees about real world effects [20]. When credible structure exists, actions are restricted to controllable ancestors of Y and graph consistent feasibility is enforced. Practical recipes for intervention feasible recourse provide sufficient conditions and constructive algorithms for this restriction [25].

In brief, algorithmic counterfactuals explain properties of a predictive artifact f, while causal counterfactuals explain properties of the process that produces Y. The two coincide only when the feasible set mirrors admissible interventions under a defensible structural model.

3.2.3 Algorithmic counterfactual explanations methods

We organize the algorithmic toolbox by how methods control proximity, sparsity, plausibility, and feasibility, and by the optimization structure used to search for a flip. The baseline view models counterfactual generation as constrained optimization

$$\min_{\delta} \; c(\delta) \quad \text{s.t.} \quad f\big(x+\delta\big) = y^\star, \quad x + \delta \in \mathcal{F}(x),$$

where c encodes proximity and sparsity, and $\mathcal{F}(x)$ encodes domain and policy constraints [91].

Direct optimization near the decision boundary

Methods in this family minimize a distance or cost subject to a label flip, often with weighted ℓ_1 or ℓ_2 penalties to encourage sparse and small edits. The practitioner specifies $\mathcal{F}(x)$ through bounds, immutability, and simple monotonic relations. Success depends on calibrated feature weights, well behaved local losses, and careful handling of mixed data types. Established surveys document these choices and their typical stability issues [92].

Exact feasibility with mixed integer programs

When features are discrete and continuous and feasibility must be hard, actions can be encoded as integer variables and solved to optimality. For linear classifiers with decision rule $\text{sign}(w^\top x + b)$, a minimal cost action a solves

$$\min_{a} \; \sum_{j} \kappa_j |a_j| \quad \text{s.t.} \quad w^\top (x + a) + b \geq 0, \quad a \in \mathcal{A}(x),$$

with additional binaries for categorical moves and policy constraints such as budgets or mutability. Such formulations certify infeasibility when no flip exists and enumerate minimal cost flipsets when recourse exists, which enables audit style reporting under realistic constraints [23].

Plausibility via data geometry or generative models

To avoid off manifold edits, plausibility can be enforced through the data geometry or a learned generator. A graph-based approach restricts moves to high density regions and searches for short feasible paths from the source instance to a target region where the label is $y^\star$ [27]. Let $G = (V, E)$ be a k nearest neighbor graph over observed points and the query x. Let $\rho(v)$ be a density estimate and let

$$w(u, v) \;=\; \alpha\, d(u, v) \;+\; (1 - \alpha)\,\big[-\log \rho(v)\big], \qquad \alpha \in [0, 1].$$

A counterfactual path $\pi = (x = x_0, x_1, \ldots, x_T)$ solves

$$\min_{\pi} \sum_{t=0}^{T-1} w(x_t, x_{t+1}) \quad \text{s.t.} \quad (x_t, x_{t+1}) \in E, \quad \rho(x_t) \geq \tau, \quad f(x_T) = y^\star,$$

optionally with a margin constraint $m(x_T) \geq \gamma$ where m measures distance to the decision boundary. This formalizes the choice illustrated in Figure 3.2.1: a candidate reached along a high-density path with sufficient margin is preferred over a nearer point that lies in a low-density pocket [27].

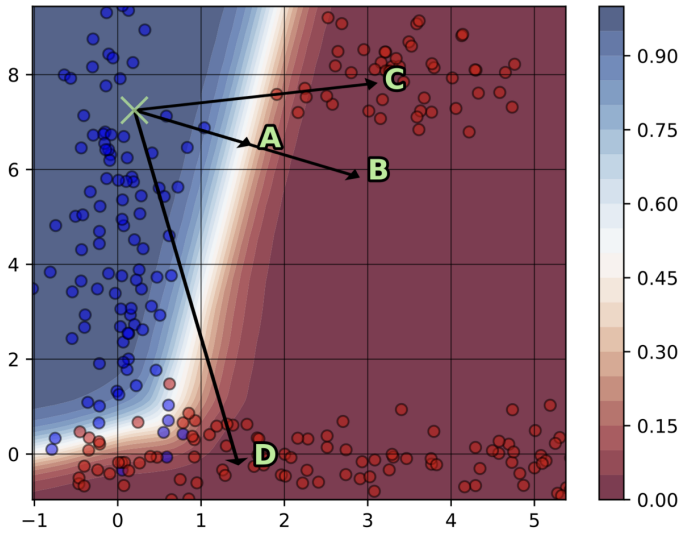

Figure 3.2.1: Feasible counterfactuals

Such feasible counterfactuals found by following high density paths in a graph over the data. Points that are close in Euclidean distance but lie in low density pockets are downgraded, while paths that remain in dense regions and end with adequate margin are preferred. In this scenario D, is the optimal choice as it handles proximity and plausibility, better than the other candidates. Reproduced from [27]

An alternative is to parameterize edits on a learned manifold. With a generator $x' = G_\theta(z)$, one can search in latent space

$$\min_{z} \; \mathcal{L}\big(f(G_\theta(z)), y^\star\big) + \lambda\, d\big(x, G_\theta(z)\big) \quad \text{s.t.} \quad G_\theta(z) \in \mathcal{F}(x),$$

which promotes realism through the generator while feasibility remains enforced at the feature or action level [24].

Diversity and robustness of solutions

In many applications we prefer to present several non-redundant options. Set valued objectives add a dispersion term to promote coverage of distinct mechanisms while maintaining proximity and feasibility. A common strategy scores a set $\{x'_1, \ldots, x'_K\}$ with a determinantal volume $\log\det K$ built from a kernel K over candidates, and balances this against loss and distance

$$\min_{\{x'_k\} \subset \mathcal{F}(x)} \; \sum_k \mathcal{L}\big(f(x'_k), y^\star\big) + \lambda \sum_k d(x, x'_k) - \mu \log\det K.$$

Robustness concerns are addressed by margin buffers or shift aware constraints that keep a solution valid under small model or distributional perturbations. Recent analyses for tree-based predictors show how sparse counterfactuals can fail under adverse perturbations and how robustness penalties mitigate this failure mode [113].

Transparent evaluation includes success rate, proximity, sparsity, plausibility, and feasibility. When sets of counterfactuals are returned, a diversity measure and per instance coverage are informative. Documentation of the cost model, the construction of $\mathcal{F}(x)$, and the background distribution used for any density terms supports reproducibility [21].

Counterfactuals become practically meaningful when edits correspond to real actions with documented burden and policy compliance. The next section develops actionable recourse[23], where the object of interest is not only a counterfactual point but an admissible action plan with certified feasibility and cost, and where local prescriptions can be aggregated into global, human interpretable summaries [28].

3.3 Actionable Recourse

Actionable Recourse is a field of tasks that utilizes counterfactual explanations, in a manner of exploiting the wrongful way a predictive model has been trained, detecting bias, and in specific settings providing real world advice to individuals to succeed on a certain task. As a phrase, actionable recourse logically proposes the change of the direction of an individual via actions[114, 115]. The main question the task asks is, given a predictive model with fixed parameters and training, and a negatively classified individual (they have assigned the undesired label), can we recommend a feasible change to the persons' features that flips the decision of the predictor, thereby operationalizing individual agency rather than offering post-hoc rationalizations[23].

Counterfactual explanations motivate recourse by framing the smallest change that would yield a desirable outcome, but do not, by themselves, enforce feasibility or action sets [26]. This distinction motivates optimization with explicit constraints. Several surveys formalize algorithmic recourse as constrained recommendation problems that are model-agnostic in principle, yet depend on explicit assumptions about mutability, costs, and plausibility in practice. In this section, we focus strictly on individual-level recourse, and we distinguish it from global and summary approaches [116].

Formal Definition

Let $x \in \mathcal{X} \subseteq \mathbb{R}^d$ denote the features of an individual, $f : \mathcal{X} \to \{0, 1\}$ be a fixed classifier, and the desired outcome is $y^\star = 1$. An action $a \in \mathcal{A}(x) \subseteq \mathbb{R}^d$ is a feasible edit with cost $c(a \mid x)$. The recourse problem is

$$\min_{a \in \mathcal{A}(x)} \; c(a \mid x) \quad \text{s.t.} \quad f(x + a) = y^\star, \tag{3.3.1}$$

which encodes the minimal effort for feasible label flips [23]. The feasibility set $\mathcal{A}(x)$ encodes immutability (e.g., age cannot decrease), directional and conditional mutability (e.g., education may only increase), and domain constraints for discrete, ordinal, and continuous features [116]. To make costs comparable across features, a common choice is a weighted ℓ_1 metric

$$c(a \mid x) = \sum_{j=1}^{d} w_j \, |a_j| \quad \text{with} \quad w_j = \frac{1}{\mathrm{MAD}(X_j)},$$

where MAD represents the median absolute deviation of a feature x_j defined as

$$\mathrm{MAD}(X_j) = \mathrm{median}_i(\, |X_{ij} - \mathrm{median}_i(X_{ij})| \,) \, .$$

so distances reflect relative change magnitudes rather than raw scales [23, 26].

3.3.1 Approaches of Actionable Recourse

Several methodological approaches implement the same task, however, we describe the four main group. Thus, in the following segment we briefly analyze *linear separator, non-linear separators and path-feasible recourse,*. In order to be more specific we have to examine exact actionable recourse for linear models.

we use Integer Programming and Flipsets. For linear separators denoted as

$$f(x) = \mathbb{1}\left[w^\top x + b \geq 0\right],$$

we fix the parameters and the training data. The actionable recourse question asks whether a negatively classified individual can achieve a favorable decision through a feasible minimal cost change to the features while respecting mutability limits, domain bounds, and discrete encodings. This setting admits an exact mixed integer formulation that imposes the simple flipping requirement

$$w^\top (x + a) + b \geq 0,$$

The formulation represents categorical and ordinal edits with binary variables and enforces feasibility with linear constraints, so the solver either returns a minimal cost action or certifies that none exists. For small to medium feature spaces, this yields individual-level solutions with provable optimality, and it does so without altering the underlying model. To move beyond a single prescription we need to define the flipset. This is formalized as the collection of distinct minimal cost actions available to the same individual, with each action changing a different subset of features while still flipping the decision. A flipset is useful because it offers real alternatives that may differ into real-world projection or risk, even though each is optimal on its own support.

We construct the flipset by solving once to obtain the initial minimal action then adding an exclusion constraint that forbids the support just used and resolving to find the next distinct minimal action. We repeat this process until no further feasible option remains, or a stopping rule is met. The same rational matches audits at a population level, since per individual outcomes can be aggregated without retraining the model. For these runs it is straightforward to report feasibility rates and distributions of recourse costs, while the structure of the flipset helps reveal heterogeneity in available actions and possible system barriers. Taken together the exact formulation and its flipset extension provide prescriptive guidance for individuals and a transparent basis for formal auditing, mainly used in corporate or institutional applications[23].

On the other hand, recourse applies on model-agnostic and non-linear settings of classifiers. The objective here is translated to turn the problem into logic constraints and solving them exactly when the theory is decidable. For this matter we need to define sound and complete SAT/SMT search. Specifically, Satisfiability Modulo Theories (SMT) extends the already known SAT problem that solves propositional satisfiability with richer theories, such as linear real arithmetic. A solver is defined as sound if it never returns false positives and is defined as complete if it returns a solution whenever one exists. A theory is decidable if the solver terminates with a definite yes or no answer. A general model agnostic template is to encode the predictor as a theory $\Phi_f(x + a)$ and to encode cost as $\Phi_c(a)$. Then we encode feasibility $a \in \mathcal{A}(x)$ and solve

$$\min_a \; c(a) \quad \text{s.t.} \quad \Phi_f(x + a),\; \Phi_c(a),\; a \in \mathcal{A}(x)$$

with an SMT solver over decidable fragments. This supports discrete variables and non-differentiable costs while preserving formal correctness [22]. This logical view sets up a unified language for additional properties, notably diversity and plausibility, which we address next.

We now have to address diverse and coherent counterfactuals under discrete logic. Single-headed recommendations are often insufficient in practice because users and auditors benefit from multiple realistic alternatives that remain consistent with domain logic, for example valid one hot encodings. A mixed-polytope encoding operationalizes such discrete constraints so that we can explicitly search for diverse yet coherent sets of actions improving user choice and coverage under the same formal appliance [23]. Ensuring logical coherence is necessary but not sufficient recommendations should also be plausible with respect to data which motivates data driven constructions.

The final approach refers to path-feasible recourse that constrains the actions to stay on-manifold. This significantly shows similarity with FACE algorithm we presented in the previous section(3.2.3). In order to implement such methods, we construct a neighborhood graph over the observed data, such as a k-Nearest Neighbors (k-NN) graph in an appropriate metric space, and restrict candidate edits to paths that remain on this graph, so that each small step respects the feasibility set $A(x)$. This optimization target is a minimum cumulative path cost from the current instance x to any node that the black-box f labels as favorable. The output is a sequence of small, interpretable edits rather than an off-manifold jump. Next, we treat these graphed-path methods as plausibility devices that complement exact optimization for parametric models, for example, Mixed Integer Linear Optimization (MILO), rather than replace them. The graph constraint enforces support awareness, while MILO delivers cost minimality and infeasibility certificates under explicit cost-immutability rules. Full stop [27].

3.3.2 Feasibility, Plausibility, Robustness, and Evaluation

To complete the discussion on the concept of Actionable Recourse completely, we address four primitives that make the previous methods operational. These aspects are *feasibility, the cost functions, the plausible device,* and the *robustness and evaluation protocol*, which are necessary preliminaries of the comprehension of the following matters. Our aim here is to remove ambiguity in what actually counts as a permissible edit, how effort is measured, how proposed actions that provide counterfactuals out of distribution are ruled out, and finally how stability can be quantified under predictive multiplicity as far as the variety of predictions of different under the same action is concerned.

To begin with, we restate the feasibility set $A \subseteq \mathbb{R}^d$, which encodes immutability of coordinates of instances, via hard constraints, directional allowance, and conditional constraints that allow features to be changed depending on specific settings of the domain. It is important to note that the feasibility set has to be specified before any solver is invoked and has to be unchanged during the process of training, auditing, and robustness analysis. Otherwise, it is almost guaranteed that the results and actions given will have wrongful edits, which if one is not experienced enough, can be attributed to the faultiness or malfunction of the predictive model [116]. Secondly, a plausibility device is required to prevent off-support edits. This ensures that, when comparing methods, we have a fixed constraint to keep the actions proposed have the same boundaries and restrictions, like minimum density thresholds for any proposed counterfactual.

Subsequently, the need to compare efforts across heterogeneous features and instance vectors, has led to the employment of population calibrated cost functions, that similarly to the previous properties, should be fixed in all reports. Common, cost functions are represented by the original weighted norm functions and derived distance functions, that are however, always dependent to the scale of the features, so the dimension of each feature will impact in a disproportional and unfair amount, in the total cost a move from the original point of feature space to the target. Such heuristics are avoided in the task, while one can argue that by a robust analysis of the domain examined each time on a dataset, the heuristic cost functions with the correct adjustment of weights will provide a sufficient evaluation of effort. However, recent works have proposed several functions that are scale-invariant and can proportionally and fairly encode the importance and the nature of each feature. The notable example that will be closely examined in the rest of this thesis is the *Maximum Percentile Shift* (MPS) function that was introduced as the audit function by [23]. The equation is defined as

$$\mathrm{MPS}(a \mid x) = \max_{j \in S(a)} |Q_j(x_j + a_j) - Q_j(x_j)|, \tag{3.3.2}$$

where Q_j represents the empirical Cumulative Distribution Function over the distribution of feature j, and $S(a) = \{j : a_j \neq 0\}$. The use of this cost function will be analyzed in detail on Section 3.4 and Chapter 4.

Robustness is a needed primitive to evaluate the process of Actionable Recourse, as models with comparable performance and risk, can induce different decision boundaries in the feature space. This leads actions that succeed for one fit of a model, fail for another. This urges the adoption of another fixed protocol. Some proposed methods introduce the construction of an ensemble $\{f^{(b)}\}_{b=1}^{B}$, that operates via refits of the algorithm that samples perturbations. For each b, the computation of the minimal cost action $a^{(b)} \in A(x)$ and the recording of the flip success, summarize through an aggregate score. In methods like linear models, a margin buffer can be used to strengthen the flip constraint over a specific threshold of success. Throughout, the feasibility set $A(x)$ and the cost function must remain fixed, so that instability can remain only variant to model variation rather than changing constraints[117].

Finally, the most common evaluation metrics, that are used in order to compare results of different works, are the necessary common ground the methods should share to understand each method strengths and weaknesses. The foundational standardization fields are, *flip/success rates, feasibility rates, distribution aware minimal costs, sparsity constrained actions, robustness descriptors,* and *diversity* when enumerating multiple minimal or near-minimal actions, such as the number of distinct flipsets and their feature overlap. These metrics computed over the same action set, the same cost function and the same plausibility device, ensure a coherent and reproducible narrative as stated by [116].

All in all, we understand that actionable recourse has a very strict methodology, yet if the setting of examining a problem or a domain of interest are strictly and carefully stated and justified, the utmost goal of comparison between works is achieved. As a task of explainability, recourse, can be argued that it is a task that might

have a post-hoc character, yet it yields answers that substantially can interpret a model's decisions. While decision boundaries across models vary, similar actions may be proven to have the least cost needed to flip a label for an individual with the same scores across models. However, this uncertain and slightly ambiguous part of the process, has lately been attempted to be faced by various methods that handles the task globally by partitioning the population. In the next section we discuss the second part of related work of our thesis, that of Actionable Recourse summaries, and how methods try to collectively extract explainable actions for affected populations.

3.4 Related Work - Actionable Recourse Summaries

The task of Actionable Recourse has shown to have a high adaptation by application fields that need to provide such explainable and audit processes to populations, either for research reasons or practical considerations, such as the regrouping and reforming of a company's structure and hierarchy of employees. For such domains where global knowledge is more valuable than specialized over individuals, the need for evaluating recourse across correctly selected groups is highly demanded. This approach has the advantage of providing a shared pool of choices to individuals, from which the optimization problem of actionable recourse is generalized and globally computed for a whole population of individuals. This has seen to provide great improvements as far as the generalization of actions is needed, and the economy of less personalized modifications for individuals.

While in late years the literature has grown on globally examining counterfactual explanations and utilizing them to extract actions for actionable recourse tasks, every approach has a distinct common ground to compare with. In this section, we review two global recourse summary formalisms. *Counterfactual Explanation Trees* (CET) provide a single, consistent action per partition cell (leaf) with an explicit tree structure. *Actionable Recourse Summaries* (AReS) provide two-level rule sets that summarize recourses for user-specified or discovered subgroups. Nevertheless, recent valuable works, that extend these methods exist like GLOBE-CE[118], or GLANCE-T and GLACNCE-C methods by [119], however in the thesis timeline we are not able to further analyze these works or compare with them. For this reason, we establish the foundations as presented by the following methods.

3.4.1 Actionable Recourse Summaries - AReS

In the AReS method, the authors present a global summary as a small collection of triples $R = \{(q, c, c')\}$. Each triple states who the description applies to, what the current situation is, and what changes are recommended. Specifically, q describes a subgroup using simple checks on features, c matches the current state for those cases, and the part c' gives the concrete edit to apply, meaning the action to apply on instances. In plain words, we describe this as *if an instance satisfies q and also matches c, then the advised change is* c'. The set R of such triples is kept short so that a reader can go through it easily, understand the proposed edits, and personally evaluate their choices and have their verdicts, based on 10-15 rules per set [28].

Problem setting

To formally address the matter the authors denote X_{aff} as the instances with an adverse outcome, also referred to as the affected population by the predictive models' decisions. Then, as B they denote the fixed predictive model used, and R is the candidate rule set that contains all the optimal and feasible actions proposed.

$$\text{incorrectrecourse}(R) = \sum_{i=1}^{M} |\{x \mid x \in \mathcal{X}_{aff}, x \text{ satisfies } q_i \wedge c_i, B(\text{ substitute } (x, c_i, c_i') \neq 1\})| \tag{3.4.1}$$

Equation 3.4.1 shows how often a proposed edit fails to flip the undesired label, which was decided by the given predictor B, when it was applied as an action.

The next metrics presented by the authors have a greater significance on the quality of actions over the population of instances they are applied to. The coverage metric 3.4.2 shows the impact of the final rule set on all instances of the affected population. This impact is quantified by the percentage of instances that were given an applicable action, meaning an action that is specified for their needs based on their features and the if-then-else rules partitioning of the dataset. Also, an action is applicable on instances iff the edits proposed for an individual are vectors that prescribe change from the original instance's feature values, meaning that at least one coordinate of the state c' is different from the respective feature values at state c. Thus, coverage determines how many individuals received an actual action proposition and whether it was applied to them.

$$\text{cover}(R) = |\{x \mid x \in \mathcal{X}_{aff}, x \text{ satisfies } q_i \wedge c_i \exists I \in \{1, \ldots, M\}\}| \tag{3.4.2}$$

On the other hand, the frameworks' cost functions quantify how demanding the prescribed edits are over the entire recourse policy R. The formulation introduces two aggregate quantities:

$$\text{featurecost}(R) = \frac{1}{M} \sum_{m=1}^{M} \text{cost}(c_m'), \qquad \text{featurechange}(R) = \frac{1}{M} \sum_{m=1}^{M} \text{magnitude}(c_m'), \tag{3.4.3}$$

where M is the number if rule recommendations under consideration. In practice, M can be instantiated either as the number of unique prescribed actions $c_m' \in R$, or as the number of *applicable assignments* of these actions across covered individuals when evaluating on a dataset. These aggregates are used to punctuate the "typical burden" of the provided recourse across the population that is the one intended to receive it [28].

Cost c_m' is a domain-informed scalar that quantifies the effort of carrying out the prescription c_m'. This cost is not restricted to an ℓ_p-style norm, as it can incorporate heterogeneous factors, such as temporal or structural difficulty costs, that evaluate how long will it take to achieve a goal or how difficult it is to increase your hierarchical status.

On the other hand, the magnitude(c_m') measures the size of the modification in feature space, interpreted as the total absolute size of feature edits required by c_m'. This applies to how many coordinates of the feature vector change and how much if they do so. This quantity indicates how invasive the action is with respect to the current state c_m.

Having explained the costs, their aggregates serve indeed different audit questions. The first featurecost(R) provides the answer to, *on average, how demanding are the actions proposed to a population, according to the*

context of the domain. For featurechange(R) the explanation offered is, *on average, how large are the edits on the feature space.*

This separation of costs matters, as two actions can have the same magnitude, yet radically different cost. This lies on an audit example of income increase, where the action proposes to two individuals a raise of income of magnitude of 10%, while the cost of such a move, might be less demanding for one with low income, the same raise will have a greater cost for a person with significantly higher payroll. For this reason, the need to report both effort functions is necessary, to avoid semantic burdens and express geometric distance truthfully, while preserving the ability for comparison when applying recourse to different populations.

The framework then, explicitly treats interpretability, as a resource that can be constrained. This is meant to provide a clear, readable, and comprehensive output for the user to read. To achieve this qualitative aspect, the following structural complexity measures are defined:

$$\text{size}(R) = |R|, \quad \text{maxwidth}(R) = \max_{(q,c,c')\in R} \#\text{predicates in}(c \wedge q), \quad \text{numrsets}(R) = |\text{rset}(R)|. \tag{3.4.4}$$

Here text(R) represents the length of the summary, so that the fewer rules exist, the easier it is going to be to read the artifact and communicate the results for the users. Then, the maxwidth(R) is a measure of local complexity, by counting atomic predicates in the most complex prior rule of form $c \wedge q$. This way the authors bound worst-case semantic loads for every rule. Finally, the numrsets(R) expresses fragmentation of the summary, as the number of different recourse sets, which is needed to avoid the generation of near-duplicate rules or over-fragmented ones that are too sparse to have a practical effect and provide explainability.

Optimization problem

The above measures and metrics work together so that AReS can return the most suitable candidate R. The goal is to find a small set of triples that covers as many affected individuals as possible, recommends realistic and practical edits, and does so at minimal cost. To make this feasible, the search space is discretized into bins. While this allows the algorithm to operate on structured data, it also makes the space extremely large and difficult to handle for larger datasets. Still, the algorithm improves its search step by step, narrowing the space and enforcing limits on size and rule width, which helps it get as close as possible to the best achievable values.

Through experimentation and careful analysis, it becomes clear that the method's main limitation is its reliance on approximation rather than on an exact solver. In other words, the solution it finds is near-optimal but not guaranteed to be strictly optimal. Another limitation is that the combinatorial space of possible rule combinations is only partially explored, as the algorithm restricts some combinations to keep the process manageable. Scalability is also an issue, as when the framework is applied to larger datasets, the runtime quickly becomes impractical on standard hardware.

That said, it would be unfair to dwell only on these shortcomings. AReS, together with the CET framework discussed in the next subsection, played an important role in shaping the foundation of this thesis. Both works provided valuable insights into how to design, optimize, and evaluate actionable recourse summaries that balance interpretability with practicality.

3.4.2 Counterfactual Explanation Trees - CET

Counterfactual Explanation Trees (CET) holds a special part in this thesis. This work has highly influenced the subsequent work of this thesis. Specifically, in the thesis, we formulate the same problem setting provided by CET, and formulate it through a different framework and methodology. However, almost everything we discuss below that refers to CET is applied and adapted in our later described framework in chapter 4.

CET was the first work to instantiate the task of actionable recourse summaries through tree structures, thus giving a new, more transparent structure of the summary that follows specific hierarchical rules via root-to-leaf paths. The key aspect of understanding the framework's results, is that through the partitioning of the feature space that is handled by the decision tree, each leaf assigns a specific action to a subpopulation. The choice of action is done through the basis of actionable recourse, meaning the final chosen action per leaf minimizes the cost of the move of instances that need recourse, while ensuring the chosen actions also, maximize the success of flipping the label when the edit is applied.

Setting

The authors use the same problem setting, of having a fixed classifier $f : \mathcal{X} \to \mathcal{Y}$ that provides predictions on a certain dataset D. Then the target population is reduced to instances x such that their predicted label is different than the desired class, thus creating the affected population that will be used for recourse. Without loss of generality, the declare as the action set of all possible feasible actions to be applied on instances, to have a global scope and defined as $\mathcal{A}(x) = \bigcap_{x \in \mathcal{X}} \mathcal{A}(x)$. This denotes the intersection of the feasibility set for every affected individual. Here, however, for practical considerations, the intersection of subsets is implemented separately for each leaf node to reduce computation time and memory load.

The optimization problem follows three hierarchical equations, which follow the order from instance-wise optimization, to leaf optimization, and conclude in the final tree solution optimization. The metric of evaluation instance-wise is defined as the invalidity score that is described as the cost of an action a over an individual x and the addition of the scaled version of the ℓ_{01} loss function that returns 0 in case the action successfully flipped the label or 1 if it didn't achieve its goal.

$$i_\gamma(a \mid x) = c(a \mid x) + \gamma \cdot l_{01}(f(x + a), +1) \tag{3.4.5}$$

The scaled version of the loss function is achieved by the γ parameter, which is a positive scalar, and its use is to evaluate the significance of the success of flipping the label for the specific setting for the domain the task is used. This makes it a trade-off parameter, that if chosen to have a value between $(0, 1)$, it reduces the flip loss, thus influencing the cost function to have higher influence on the decided actions that are returned on leaves.

This leads to leaf evaluation of the score that is presented as an aggregate of equation 3.4.5 and defined as

$$\text{minimize}_{a \in \mathcal{A}(X)}\, g_\gamma(a \mid X) := \sum_{x \in X} i_\gamma(a_i \mid x). \tag{3.4.6}$$

The leaf-wise optimization is the minimization of average invalidity on the subpopulation X of the candidate leaf, such that the optimal action $a^\star \in \mathcal{A}(X)$ is returned as an outcome. The problem of leaf optimization has a local character because even though it optimizes through the average score of the subpopulation examined, it does so by heuristically creating candidate actions that are then included in the action set used for the leaf optimization. The authors create actions through a variety of approaches, but the one they propose is based on LIME approximations. Then this optimization problem solved as an MILO formulation and the use of state-of-the-art solvers of this nature as IBM C-PLEX[1] or Gurobi solver[2]. Additionally, a last metric as far as local evaluation is examined, is the use of the cost function the authors use. We have previously stated that a scale-invariant cost function used in Actionable Recourse is the Maximum Percentile Shift function that exploits the effort needed to apply an edit to an instance based on the Empirical Cumulative Distribution Function. This cost function is well defined by the users, and because of its appropriate use for the task, it is later adapted by our formulation.

[1] https://www.ibm.com/analytics/cplex-optimizer
[2] https://www.gurobi.com/solutions/gurobi-optimizer

Before delving into the analysis of the algorithm and the global optimization problem, we have to note a particularly important clarification that the authors make and is needed for out further discussion and later formulation of out method in the subsequent chapter. For the objective function $g)\gamma$,the authors indicate and prove that it holds monotonicity with respect to a partition X of the dataset[1]. Specifically, they denote that given two subsetsX_1, X_2 of a set of instances $X \subseteq \mathcal{X}$, such that $X_1 \cup X_2 = X$ and $X_1 \cap X_2 = \emptyset$, the relation that holds is $g_\gamma(a^\star_X \mid X) \geq g_\gamma(a^\star_{X_1} \mid X_1) + g_\gamma(a^\star_{X_2} \mid X_2)$. This indicates that the partitioning of the input space produces actions of higher effect based on their given score.

At this point, having made clear the procedure of optimization per instance and leaf, we need to present the global objective function and the algorithm of tree learning and action assignment. The authors first define the tree solution $h : \mathcal{X} \to \mathcal{A}$, such that the decision tree h assigns an action a to a given input instance x. The global optimization objective is defined as the minimization of the average invalidity of the whole tree structure, by aggregating the invalidity per leaf and averaging by the total population set cardinality. To this objective, however, an added term is being introduced, that is, regularizing the tree based on the number of leaves it has, an important aspect when the need of the final objective is not only to minimize the cost function but have control of the complexity of the tree structure. The objective $o_{\gamma,\lambda}$ is defined as

$$o_{\gamma,\lambda}(h) = \frac{1}{N} \sum_{l \in \mathcal{L}(h)} g_\gamma(a_l \mid X_l) + \lambda \times |\mathcal{L}(h)|, \tag{3.4.7}$$

where the first term is the aggregated average invalidity over the whole tree structure for a tree h over all the leaves that belong in the leaf set $\mathcal{L}(h)$, and the second term describes the normalization factor based on a parameters λ that penalizes large number of leaves so that complexity is controllable and reduced whenever needed.

Algorithm

The Algorithm provided by the authors, named *Stochastic Local Search*, even though based on the number of iterations and, of course, the dimensionality of the action set $\mathcal{A}$, it has some limitations due to it probabilistic approach, it is claimed and justified that returns optimal results as far as the nature of the tree objective function is concerned. For completeness of the section's discussion, we briefly explain the steps of the algorithm:

1. Initialize an empty tree structure based on the dataset examined.
2. Then for a given number of iterations $t = 1, \ldots, T$, given a probability δ that is approximately equal to $\frac{1}{3}$ and some context of complexity based on the condition (I) $|\mathcal{L}(h)^{(t-1)}| \leq \frac{\gamma+\lambda}{\lambda}$
   (a) Either, edit the tree by inserting a node with a rule randomly if the $\delta \leq \frac{1}{3}$) and condition (I) is true
   (b) Or if the condition (I) does not hold and $\delta \leq \frac{2}{3}$, edit the tree by randomly deleting node
   (c) Or if none of the above conditions hold, replace a rule that had already been introduced, randomly, with another from the action set.
   (d) Then evaluate each leaf independently based on the leaf invalidity equation, and retrieve the minimal action to be chosen per leaf
   (e) Evaluate the tree's performance under the global objective function, replace the tree structure with the one of the previous iteration, if the current is not better based on the objective function
3. After the last iteration, the tree structure that is returned as the optimal solution is the one that, throughout all iterations, had the minimal score of global objective $o_{\gamma,\lambda}$.

Although the algorithm promises optimal solutions, several limitations have been distinguished. First, this probabilistic approach does not strictly provide optimal solutions. This derives from the fact that the solutions are a product of convergence after several iterations, which are set manually via experiments and observation of the behavior of the objective function as iterations increase, given the respective domain of instances, size of the training batch, and cardinality of the action set. Similarly, a closely related yet not identical limitation is the scalability issue, for exactly the same reasons that hold for the previous limitation of not

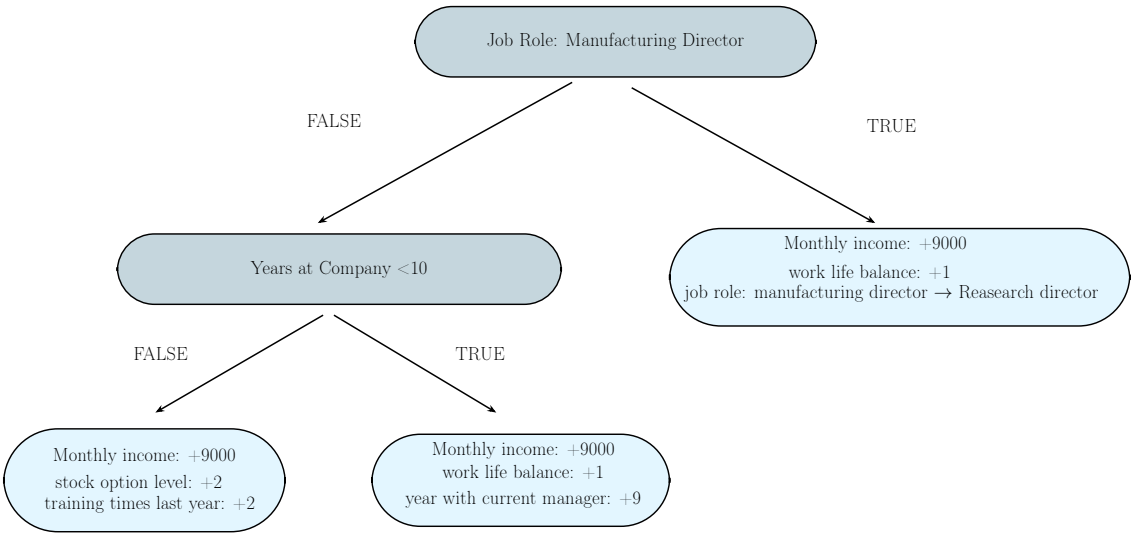

Figure 3.4.1: The transparent tree structure of CET

The specific example shows the interpretable aspect of the tree structure. Each leaf node, based on the split that occurred, thus the partition of the dataset, assigned specific actions for these groups of individuals that collectively minimize their cost and maximize the success of flipping the label when given as inference to the classifier.

proven global optimality. The high dimension of feature space burdens the tree structure mechanism to find optimal splits, even in a greedy way. The action set has high cardinality, combined with a large number of instances, and is undoubtedly a task that requires higher computational power, and even with the use of optimization solvers, like the ones we referred to before, the runtime would be infeasible on non-specialized computers and processors of advanced CPU performance. Another limitation we observe is the small number of experimental data domains, as the paper is restricted to two datasets. However, this limitation, even though it exists, can be justified because of the nature of the task of actionable recourse, as a small number of datasets that completely match this task exist.

All in all, the model formulation of CET is a work of high importance and value, which handles the problem setting with all the necessary detail. Adapting the general logic of the framework, the assignment of actions on leaves, and the same metrics, we build upon this matter and introduce our work in the subsequent chapter, explaining in detail the problem setting, the formulation, and all the adaptations from the CET method.

Chapter 4

Proposal

In this chapter, we present our framework for **Summaries of Optimal Global Actionable Recourse** (**SOGAR**), a method that generalizes existing approaches for generating actionable recourse and counterfactual explanation summaries [1, 28, 119]. Unlike prior work, which primarily relies on local heuristics or instance-wise optimization, SOGAR seeks globally optimal summaries of recourse by jointly optimizing tree structure and actions under explicit cost and recourse success objectives. The key gap in current literature, is addressed by our approach as we extend the search space of actions further than the local actions previously computed, while ensuring the solutions preserve global optimality, under the multi-objective optimization, providing a Pareto front of optimal tree solutions, that aim to minimize the global cost of actions and the loss metric that occurs after an actions failed recourse.

SOGAR builds upon the dynamic programming principles of the *STreeD* algorithm [2], extending them with task-specific objectives tailored to actionable recourse. In our setting, each leaf of the decision tree prescribes an action that flips negative outcomes to positive ones with minimal cost and minimal loss. The solver maintains Pareto-optimal trade-offs across cost and success through separable multi-objective dynamic programming, ensuring each region of the feature space is associated with an optimal and equitable intervention.

Conceptually, SOGAR bridges the interpretability of symbolic decision trees with the robustness demands of modern recourse systems. It offers theoretical guarantees of optimality for separable objectives from instance-level interventions to *group-consistent* decision policies. By organizing counterfactual actions hierarchically, the resulting models remain both interpretable and globally optimal, qualities which are essential for transparent decision-making in sensitive domains such as credit scoring, hiring, and healthcare.

Contents

4.1 Contributions

The contributions of this diploma thesis can be summarized as follows:

- We propose **SOGAR** (*Summaries of Optimal and Global Actionable Recourse*), a novel framework that extends Counterfactual Explanation Trees (CET) by embedding them into the globally optimal STreeD dynamic programming approach. The approach preserves interpretability of solutions, and simultaneously ensures global optimality, over all possible solutions given in the Pareto front, which gives an extra mile of personalization to users to decide which equally optimal solutions match the standards of their need better according to their domain and task.
- Each leaf of the decision tree is assigned a single actionable recourse, chosen to minimize the overall cost of feature changes while maximizing the rate of prediction flips under a fixed black-box classifier. This formulation provides globally consistent and transparent recommendations.
- We conduct experiments on public benchmark tabular datasets, demonstrating that SOGAR achieves lower average recourse cost and higher flip rates compared to CET and AReS baselines, while scaling efficiently to larger depths and dataset sizes.
- Our work establishes a systematic framework for exploring trade-offs between action effectiveness and interpretability through Pareto optimization, offering practitioners and stakeholders a transparent tool for actionable and justifiable recourse.
- The formalization of this process is achieved through the adaptation of STreeD framework, by instantiating a STreeD-defined separable optimization task $o = \langle g, t, \oplus, \succ, c, s_0 \rangle$, and meeting the necessary and sufficient conditions for the DP recursion formulation with precomputed scores.

4.2 Motivation and Problem Statement

Within the setting of the Actionable Recourse problem, we were influenced by works like CET and AReS to extend the logic beyond heuristic-based optimizations of leaves and rule sets. In Actionable Recourse, regulators, auditors, and institutions require more than a pointed prescription for a rejected applicant. They need a global and human-auditable summary that reports: who is asked to change and what is the target of change, and how costly and reliable those changes are. Summaries that are presented via rule sets, like the formulation of AReS, provide such a view, but have the flaw of assigning overlapping or conflicting actions and leave parts of the population uncovered, which damages transparency and consistency. By contrast, CET addresses coverage and consistency through a decision tree in which every instance falls into exactly one leaf and each leaf has exactly one prescribed action, so every individual receives an explanation, and contradictions are controlled and eliminated. However, CET is trained with heuristic stochastic local search, that is , a probabilistic edit algorithm, and MILO formulation at the leaf level, and therefore cannot certify that the resulting global policy is globally optimal or even Pareto optimal when the objective is considered as two separate optimization goals. This lack of optimality can be further explained by the iteration setup of the CET algorithm, as it does not exhaustively examine every possible leaf, sub-tree, up to the point of a globally optimal solution. Additionally, this motivates us to formulate a globally optimal structure that ensures and preserves optimality throughout the entire decision tree by using optimal decision trees, by formulating the DP approach based on the STreeD implementation. Thus, this leads us to the definition of the **SOGAR** problem.

4.2.1 Definition and Setting

We therefore define the Problem as suggested by the formal definition of sections 3.3 and 3.4 and our setting based on the DP formulation. We therefore restate the basis of the actionable recourse summaries setting, and we extend on it to analyze our formulation setting addition.

We fix a classifier $f : \mathcal{X} \to \{0, 1\}$, and a desired label denoted as $y^\star$, which without loss of generality, is interpreted in the binary classification setting, and thus is chosen to be $y = +1$, while the undesired label is $y = 0$. Then we define a population of instances $D = \{(x_i, f(x_i))\}_{i=1}^n$, and its filtered portion $D_0 = \{(x_i, \bar{f}(x_i) = 0)\}_{i=1}^n$ that represent instances that have predicted labels by the classifier to the undesired

class. Then, let an action space A that contains all possible edits on instance vectors of the input space, and we define the feasibility action set be the subset of $\mathcal{A}(x) \subseteq A$, such that every action $a \in \mathcal{A}(x)$ is a feasible action that will be applied and tested on the affected instances of the population D_0. We distinguish that the action set $\mathcal{A}(x)$ is an extended cached set, which will be further analyzed below, and contains actions based on a sparsity constraint of up to 3 edits per instance. Then, we define a proper cost function that measures the effort of changes on features, and is aware of the scale of change that an edit applies, based on the distribution values of each feature, and represent the cost function with the notation $c(a \mid x)$. To complete the definition of an actionable recourse problem, we finally, provide a flipping indicator in the form of a loss function that similarly to CET formulation adds the undesired predictions of edited instances that need to have their label predicted. The loss function is represented by $l_{01}(f(x+a), +1)$.

Our optimization task is bi-objective and it goal is to find a Pareto front T of non-dominated optimal tree solutions $\tau \in T$, such that for every tree τ, every leaf l_τ has a feasible action $a \in \mathcal{A}(x)$ assigned, that simultaneously minimized the cost and loss functions as individual objectives, so that the sum of cost scores per leaf and the loss scores per leaf, are minimized throughout the tree structure. For that reason we formulate the two objectives in our problems settings as

$$C(\tau) = \sum_{x:f(x)=0} c(a(l(x)) \mid x), \quad L(\tau) = \sum_{x:f(x)=0} l_{01}(a(l(x)) \mid x), \tag{4.2.1}$$

Thus, the Pareto front objective is translated as a vector of the two scores, denoted as

$$\min_{\tau \in T_d} (C(\tau), L((\tau)), \tag{4.2.2}$$

where T_d is the already defined space of binary tree solutions that belong in the Pareto front, with the distinct restriction of maximum depth d. For completeness and direct comparison to other methods, we finally, adapt the invalidity score on tree level as the sum of the two objectives of Pareto front vector objective 4.2.2 as,

$$I(\tau) = \frac{1}{|D_0|} \sum_{x \in D_0} \Big[c\big(\alpha(\ell(x)) \mid x\big) + \gamma \cdot \ell_{01}\big(f(x + \alpha(\ell(x))), +1\big) \Big]. \tag{4.2.3}$$

that has the same γ scaling parameter as adapted from the CET formulation, to prioritize cost or loss based on personal need of the task's context. To summarize the above setting we present an outline of the necessary primitives of our task. In the outline we present:

- a fixed classifier $f : \mathcal{X} \to \{0, 1\}$,
- a desired target label $y^\star$,
- a population of instances $D = \{(x_i, f(x_i))\}_{i=1}^n$,
- and the affected part of it $D_0 = \{(x_i, \hat{f}(x_i) = 0)\}_{i=1}^n$,
- a global feasible action set $\mathcal{A}(x)$,
- a cost function $c(a \mid x)$,
- a loss function $l_{01}(f(x+a), +1)$ as a flipping indicator,
- The Pareto front T_d of all non-dominated optimal tree solutions τ with maximum depth d
- the aggregated tree functions $C(\tau), L(\tau), I(\tau)$, that respectively aggregate action costs, flip loss, and their scaled sum.
- and the Pareto front objective in the form of the vector $\min_{\tau \in T_d}(C(\tau), L((\tau))$.

4.2.2 STreeD adaptation

Based on our analytical statement of our problem of actionable recourse, we now need to proceed by formalizing the problem and the optimization task, exactly as STreeD framework defines an optimization task, and then we continue to prove its separability, that as we already referred in section 2.4.3, is a set of conditions that are proven to be necessary and sufficient to define a new optimization task under the STreeD solver.

SOGAR as a STreeD optimization task

As previously stated in 2.4.1, an optimization task is defined by the tuple

$$o = \langle g, t, \oplus, \succ, c, s_0 \rangle,$$

where g denotes the cost function, t denotes the transition function, $\oplus$ the combining operator, $\succ$ is the dominance and comparing operator, c resembles the feasibility constraint, and finally, the task is represented by an initial state symbolized as s_0.

We denote that a state s records the instances that reach a node and the remaining depth d, and define the value space $V = \mathbb{R} \geq (0, 0)$, whose first coordinate is the sum of recourse cost and the second is the sum of instances that failed to flip their label. As STreeD labels that can be assigned on leaves, we consider every feasible action $a \in \mathcal{A}(x)$. Given this, we now map our problem settings on the optimization task definition given by the STreeD framework. For this reason, we start matching our components by the cost function g. In our task, the evaluation metric is the bi-objective vector that the Pareto front evaluates, thus, the vector of the two functions, of the cost of actions and the loss function, and we express it as

$$g(s, \alpha) = \Big(\sum_{x \in X_s} c(\alpha \mid x), \sum_{x \in X_s} \ell(\alpha \mid x) \Big), \tag{4.2.4}$$

with $l(a \mid x) = \mathbb{1}\{f(x + a) \neq 1\}$. This function mimics as previously said, the CET invalidity, without scalarizing, so that we can obtain the Pareto front solutions.

Then, the transition component is instantiated as $t : S \times F \times \{0, 1\} \to S$, given the state s and a binary split decision on feature $f \in F$, the return of the function is the next state after branching on feature f and is solely determined by the current state and splitting feature. This can be expressed as

$$t(s, f, 1) = \langle \{x \in X_s : f(x) = 1\}, d - 1 \rangle, \quad t(s, f, 0) = \langle \{x \in X_s : f(x) = 0\}, d - 1 \rangle \tag{4.2.5}$$

meaning that the two child states that occur after the split on feature f contain the instances that correspond to the 0 and 1 values of the feature f, respectively, the remaining depth is decreased by 1 and are fully independent. Thus, the transition function holds in our task.

The next component is our additive combination operator $\oplus$ that is expressed as simple element-addition over the objective vectors, $v = u_L \oplus u_R = u_L + u_R$. The dominance relation and comparison operator $\succ$ exists via our Pareto dominance operators on $\mathbb{R}^2$ for non-negative values. We express these relations as, $u \succ u'$ iff $u_i \leq u'_i$ for all i and $u_j < u'_j$ for at least one j.

The constraint c is represented by our feasibility necessity of actions, which means that every action a must be feasible for all $x \in X_s$, the minimum number of instances that represent a leaf node set to a value m, and the maximum allowed edits per action.

Finally, we initialize our problem given an initial state s_0 that represents the worst solution, which is acting and representing as an upper bound to our solution space.

This formalization allows our task to be solved based on dynamic programming recursion of the framework, preserving Pareto optimality across both of the optimization objectives.

Proof of Separability

Based on the definition of SOGAR via the STreeD primitives of what an optimization task is, we proceed to prove that our task will be solved under the framework as it is separable under the strict criteria presented in Section 2.4.3. The needed criteria are, to have a Markovian cost and transition function, a combining operator

that preserves order over the dominance operator, the constraint c is anti-monotonic and that optimality is preserved.

First, our cost function is indeed Markovian, as we let $s = \langle X_s \rangle$ denote the state with instance subset X_s. Then, for any feasible leaf action $\alpha \in \mathcal{A}_{\text{leaf}}(s)$, define the *leaf* cost vector as the equation 4.2.4, by which we obtain that $g(s, a)$ depends only on the *current* state X_s and the *immediate* decision α. For the cost function of the leaf, we consider it to be the "per objective" sum of cost and loss for an action and the state s.

Then, for a binary feature $f \in F$ and branch $b \in \{0, 1\}$,

$$t(s, f, b) \;=\; \langle X_{s,b} \rangle, \qquad X_{s,1} = \{x \in X_s : f(x) = 1\},\; X_{s,0} = X_s \setminus X_{s,1}. \tag{4.2.6}$$

The child states depend only on the current state and the feature f, hence t is Markovian.

The combining operator $\oplus$, preserves order with respect to the dominance operator $\succ$. The operator is defined as element-wise addition of cost function g. For the Pareto dominance relation on non-negative values of $\mathbb{R}^2$ addition is order preserving. Since $u_L \succeq u'_L$ then $u_R + u_L \succeq u'_L + u_R$. This ensures that combining Pareto optimal sub-trees via $\oplus$ yields Pareto optimal nodes.

The constraints of feasibility, minimum number of instances per leaf, and maximum number of edits, are anti-monotonic since any tree that does not satisfy these constraints then no tree that contains it can be feasible. The anti-monotonic constraints, and the preservation of order provide that our task satisfies the principle of optimality as stated by [2]. Having proven all the above, the SOGAR task meets the criteria of being separable, as the cost and transition functions are Markovian, by only depending on their current state, and our task satisfies the principle of optimality.

DP recursive relation for SOGAR

Thus, our task is able to be handled by the solver, because each subsolution is optimal independently and does not minimize its objectives based on context to other sub-trees. Thus, we can achieve the global optimum via the Pareto optimal solutions. Our task, task now is eligible to be expressed by a recursive relation based on DP and the framework's DP relation. Specifically, we express the global optimal solutions as

$$T(s, d) = \begin{cases} \operatorname{opt}\big(\{g(s, \alpha) : \alpha \in \mathcal{A}_{\text{leaf}}(s)\}, s\big), & \text{if } d = 0, \\ \operatorname{opt}\Big(\bigcup\limits_{f \in F} \operatorname{merge}\big(T(t(s, f, 1), d-1), T(t(s, f, 0), d-1), s, f\big),\; s\Big), & \text{if } d > 0. \end{cases} \tag{4.2.7}$$

As previously explained in section 2.4.3 and by the general recursive equation 2.4.9 of state and depth, the equation returns all feasible, non-dominated objective vectors that any sub-tree of remaining depth d rooted at state s can achieve. Here at depth $d = 0$ the return of the algorithm is all the optimal leaves, because remaining depth equal to zero resembles leaf nodes. On the other hand the description of the second segment is the examination of an internal or root node, for which we consider each possible split f that splits in two children. Then, for each child, we get the Pareto set of optimal solutions recursively. This leads to memorization of solutions that are independently optimal, and thus, we form all possible combinations of left and right children to form larger trees. The union indicator suggests that we want to collect to our final solution set, all possible combinations by some split on every node. The combination of both those segments of the relation provides the entire Pareto front of independently optimal solutions.

4.3 Technical Approach

We continue the discussion of our task by presenting our more technical, yet important details. In this section, we discuss our choice of cost and loss functions, the choice of hyperparameters and parameter tuning of the model, the Pareto front implementation, and techniques for a viable and solvable solution space. Finally, before our model's description, we analyze our method of using precomputed costs and flip rates for our cached action set.

4.3.1 Cost and Loss function

For our model, we faced a dilemma when selecting the cost function and the loss function. The loss function was straightforward to choose because the formulation used in the CET line of work is simple, comprehensible, and effective at indicating whether an instance flips after applying an action. Thus, our flip loss is

$$\ell_{\text{flip}}(x, a; y^\star) = \mathbb{1} f(x + a) \neq y^\star. \tag{4.3.1}$$

We adopt this indicator loss to evaluate flipping success similarly to [1].

The cost function required more consideration. We chose the maximum percentile shift, as used in CET, because it is scale-invariant with respect to feature distributions and yields a realistic depiction of effort across heterogeneous features:

$$c_{\text{MPS}}(x, a) = \max_{j \in \mathcal{A}} \left| Q_j(x_j + a_j) - Q_j(x_j) \right|, \tag{4.3.2}$$

where (Q_j) is the empirical cumulative distribution function (CDF) of feature (j), and $\mathcal{A}$ denotes the set of actionable coordinates.

The literature on CET and on algorithmic recourse for linear models also proposes the total log percentile shift,

$$c_{\text{TLPS}}(x, a) = \sum_{j \in \mathcal{A}} \log \left(\frac{1 - Q_j(x_j + a_j)}{1 - Q_j(x_j)} \right), \tag{4.3.3}$$

which emphasizes that equal percentile gains become progressively harder near the upper tail. This function is useful when constructing flip sets because it biases solutions toward combinations of edits that are comparatively easy within the target population, as presented by [23]. In summary, TLPS is better aligned with building flip sets, while MPS is well suited for population level audits because it communicates the minimal percentile movement required to achieve a flip in a scale-free way. Therefore, for auditing tests we use MPS, and our cost function is given by the expression above. Thus, our choice of MPS lies with its simplicity for reproducibility and for comparison with our related works.

4.3.2 Parameters and constraints

One aspect that the framework is flexible upon is the personal choice of parameters, the choice of *maximum depth* d, the *minimum number of instances contained in leaf* m, the *maximum number of internal nodes*, contribute to the personal choice of the user to where they want to extend their solution search. By keeping personally choosing the right amount of leaf size the user can produce balanced trees that generalize more than sufficiently on test sets.

However, the choice of deeper trees highly burdens the solver, especially while keeping the leaf size small, as the search space exponentially grows to infeasible runtime. For such a scenario, the framework encourages the user to use a *timeout* parameter that terminates the search process and returns the globally optimum until this point. This is of course a limitation of such frameworks, however, for our task and the datasets we examine, such deep trees might be unfaithful to generalization, as too specific actions that would target a very small partition of a population, would conflict with the tasks advantage to generalize.

4.3.3 Pareto front implementation

The Pareto front is constructed by first declaring the task is of *partial order*. To implement the Pareto front, we use the dominance functions provided by the STreeD framework and apply ϵ constraints to introduce tolerance in pairwise comparisons. This is necessary because action costs are real-valued and represented as floating point variables. Without an epsilon constraint, the search space would expand significantly, and results would not be obtained in reasonable time.

Another notable aspect of the framework is the use of upper and lower bounds during the computation of the Pareto front. These bounds prune the search space early by tightening the region of interest, so that any solution that is provably worse than a current bound is eliminated immediately without further consideration.

The size of the Pareto front is configurable through a parameter that limits how many non-dominated solutions are retained.

Specifically, for two objectives f_1, f_2 that we aim to minimize, we use an ϵ relaxed dominance defined by

$$x \prec_\varepsilon y; \Longleftrightarrow; \big(\forall i \in 1,2\colon f_i(x) \leq f_i(y) + \varepsilon\big) \ \wedge \ \big(\exists j \in 1,2\colon f_j(x) < f_j(y) - \varepsilon\big) \tag{4.3.4}$$

The framework also provides "merge and add" functions that operate after computing optimal leaves. These functions enumerate the feasible combinations of sub-trees induced by the optimal leaves, retain only those combinations that can belong to the ϵ non-dominated set, and discard any alternative that would burden the search without improving the Pareto front.

4.3.4 Binarization and Caches

The framework has a great restriction of examining and solving the optimization problems over fully binarized datasets. This restriction, however, is what justifies its advantage to provide optimal solutions for total and partial order, in high performance fashion. This, case was troublesome, as the preprocessing of the data for our task should be spit in several parts, in order to first binarize accordingly to the feature space, without extending it to dimensions where information from the structure of the dataset is distorted. Thus, in order to achieve such a goal we broke down the process of data handling in the following parts:

- A careful binarization of the original dataset, without distorting information based on unrealistic dimensionalities
- A creation of a context file that preserves information between the binarized dataset and the original dataset, also containing statistical values, such us the density and frequency of bins, to retain control over the extent of binarization
- A cache that contains all possible actions created based the feasibility criteria of the user, and the fixed constraint of maximum 3 edits allowed per action.
- The creation of a cache that includes all the precomputed costs and predictions, per instance for every action in the action set.

Specifically, referring to the cost and predictions cache, our initial approach was to compute the costs and predictions as the solver was running and optimizing the tree solutions, this was computationally infeasible, as the need to use combinations of actions would stall the processor. The easiest and most efficient approach was to predetermine all costs and predictions of the classifier, for every instance of the affected population and every action of the instance space. This might seem computationally impossible as we scale towards larger datasets. However, the computation of cost is done once for all atomic actions (one edit action vector), and then via the MPS function it was practically another $O(1)$ read from memory. Similarly, the batches of instances that were eligible for prediction, would have their prediction cached after every action was applied.

4.4 Proposed Method

These lead us into the formal description of our method. We propose Summaries of Optimal and Global Actionable Recourse (SOGAR), which is a formulation of the actionable recourse summaries problem, by exploiting the DP framework of Optimal Decision Trees of Separable Trees with Dynamic Programming (STreeD) [2].

Having already stated the problem setting, we now only describe how the process is executed in practice. We analyze how the process is divided into three sections that fully interact during the process, the workflow of the pipeline, end-to-end from the original input to the final output of the process. The pipeline is visualized in figure 4.4.1.

The pipeline, as shown in the figure, is divided into three components only for better distinction of the steps of the process, and ease of comprehension of users. Specifically, the first component refers to the training process of the classifier, the retrieval of the datasets with predicted labels, and the later inference on the trained model for instances that are edited after an action application. The second component is the

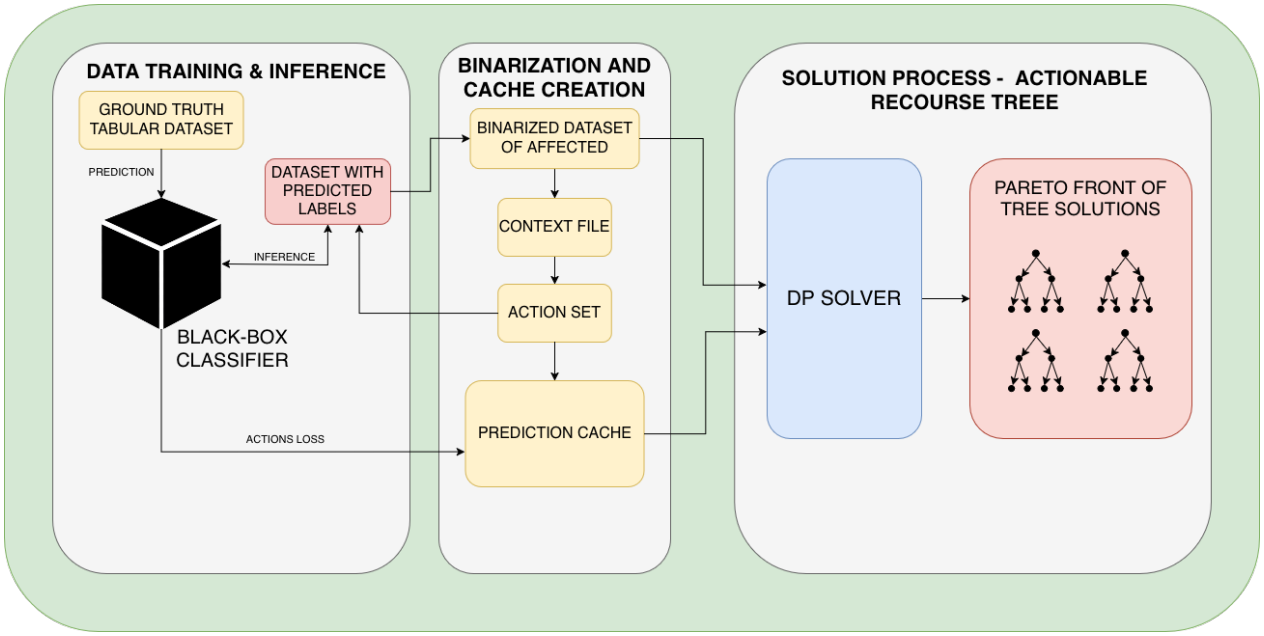

Figure 4.4.1: SOGAR Process

In the diagram of the SOGAR process, we observe the division of the process into three distinct sections, which are fully interactive during each process.

preprocessing of the predicted data into a fully binarized dataset, compatible with the DP solver, the creation of the metadata file, the action set, and the computation of the cache scores. Finally, the third component encloses only the solution process and the Pareto front solution of the STreeD algorithm, given the input of the binarized dataset, the context files, and the precomputed scores retrieved by the external cache. Below,

we unravel the description of the process into ordered, discrete steps:

1. First, we **Train and predict** on the original dataset. We fit the fixed black-box classifier f on $\mathcal{D}_{\text{tr}}$, then predict on $\mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{te}}$. This produces a post-prediction dataset with assigned labels $\widehat{y} = f(x)$. We extract the affected population $\mathcal{X} = \{x : f(x) = 0\}$, which is the only set considered in downstream recourse computation.
2. We continue by **Binarizing for tree search.** We preprocess the post-prediction dataset to a binarized representation $\mathcal{D}_{\text{bin}}$ (one-hot/thresholded features), preserving the instance indexing of $\mathcal{X}$. This step prepares the feature space for exact optimal-tree search while leaving the original (real-valued) features untouched for action evaluation.
3. Then we extract **context file**. Specifically, as a by-product of binarization, we write a *context file* $\mathcal{C}$ that records the mapping between original and binarized features, domain bounds, mutability constraints, and any per-feature statistics needed by cost/evaluation. This context file preserves the necessary connection between the original dataset values and our binarized, and is crucial for all the next parts of the process.
4. The next step, is the **generation of the action set.** Using $\mathcal{C}$, construct per-instance feasible actions $\mathcal{A}(x)$ with the user-imposed feasibility rules, for example, the immutable features excluded, and the maximum number k of edits (in our experiments $k \leq 3$). The actions based on the context file preserve the rules of direction set per feature, and each action is defined on *original* feature coordinates and is intended to be evaluated against f.
5. At this point, the **precomputed of the cache of costs & outcomes** if eligible computation. For every $x \in \mathcal{A}$ and $a \in \mathcal{A}(x)$, compute once:

$$\text{cost}(x, a), \qquad \text{flip}(x, a) = \mathbb{1}\{f(x + a) = 1\}.$$

Store these, together with action metadata, in a cache Cache. The cache is keyed by (x, a) and avoids recomputation during tree optimization; it also fixes all stochasticity before the search.

6. We, then, proceed to **Solve the tree of optimal actions.** For this part of the process, we provide $(\mathcal{D}_{\text{bin}}, \mathcal{C}, \text{Cache})$ to the STreeD solver together with structural constraints of maximum depth d, minimum leaf size m, the constraint on the Pareto front size c_p. Then, the solver assigns a single action to each leaf and optimizes a separable, leaf-additive objective under these constraints. Multi-objective search is handled via ε-relaxed dominance and bounding throughout the DP.

7. Finally, the **Pareto set** is returned. The solver outputs a Pareto front $\mathcal{T}^\star$ of non-dominated trees. Each tree partitions $\mathcal{X}$ without overlap and prescribes one feasible action per leaf, together with aggregated cost and flip statistics. These are the final SOGAR summaries.

The first advantage of our approach, which makes it stand out compared to related methods, is that the DP approach delivers the optimal solution in a totally deterministic manner because of the fixed reduction order. It implements a fully exhaustive search of all possible scenarios, creates all combinations of optimal leaves, and returns all the non-dominated trees up to the given restricted maximum depth.

In addition to that, our method exploits the flexibility offered by the STreeD framework, such that we cache the solutions during the process, for faster look-up on states that have already been examined, and uses the in full extent the upper and lower bounds that tighten the search space, on our personal request, yet still ensuring that the optimal trees will be found in deterministic time. The two precious benefits fully comply with the high-performance claims of the DP methods, and specifically the exceptional computational time of STreeD, thus aiding our method to surpass previous methods not only in improved performance and the guarantee of finding all optimal solutions, but also the great reduction of time needed for such demanding computations.

Finally, another aspect we need to address, even though we have not given enough emphasis on it yet, is the extraction of a set of solutions per run of the algorithm, and not a single one. This might be interpreted as a labyrinth of choices by some, yet we claim that such an output is more than helpful for such tasks as the recourse of population and the explainability of the black box model. To address this, the number of Pareto front solutions can be restricted by the user, which, in other words, means that it is fully personalizable to what extent that the user decides the extent of the search space, the structure of trees, their interpretability, and their complexity. As we have previously said, in section 2.4.2, the Pareto front can be tailored without loss of generality or distortion of results, based on some clearly stated and justified rules.

In our implementation, it is clear that we use a hybrid approach of the ϵ tolerance and the weighted cost function. The ϵ on boundaries and dominance operations, such that we compare solutions up to a six decimal digit accuracy ($\epsilon = 10^{-6}$), such that we can distress the solver from the computational effort needed, and thus, provide solutions with a personally chosen and justifiable accuracy of comparisons. On the other hand, the cost function uses the scaling factor γ that, based on the appropriate choice of value, reports back solutions that are optimal given the importance of the task if the objective that calculates the cost of actions. By diminishing one of the two objectives, thus scaling in to approximately 0 values, we state that we tolerate flip failure and approach the optimization problem with a high importance on the cost function. This, practically refactors the search space to prioritize the cost objective, and thus the Pareto front is restricted to one a very small portion related to the extent of an non-scaled bi-objective where the Pareto front has the "freedom" of extending across a the non-dominated curve, that connect the extreme points[1] of the solution space. On the other hand, the larger the value of γ is, the higher the importance of returning favorably effective actions is. This calibration parameter, adapted by [1], thus, offers full control over the adjustment of the search space according to the setting of the recourse task and the specialty of the domain.

Also, in a more practical view, the multiple solutions, offer the user a greater pool of choices that match the criteria of certain populations, allowing for targeted decision-making across different demographic or operational segments. For instance, one tree from the Pareto set, may prioritize low-cost and high feasibility, suitable for large populations with limited resources, while another may emphasize a higher flip success, for

[1]The extreme points are denoted by the two non-dominated solutions, that are described as having one objective coordinate as the global minimal value over the plane, and the other objective coordinate has assigned the worst value based on total ordering of this objective. Such extreme points are denoted as: (C_M, L_m) and (C_m, L_M), where the indicators M, m represent the maximal and minimal values given the coordinate.

groups that are characterized by higher risk. Empirically, the multiple choices and the visualized structures, provide a flexibility to align recourse policies given the human factor of rationalizing the proposed actions. The same aspect of multiple trees provides a great guide for researchers who aim to understand model biases or odd decisions, thus being able to have more than a sufficient amount of sample solutions to evaluate such tasks. In other words, multiplicity of solutions does not complicate the decision process but rather empowers it, transforming the optimization process to a controlled and transparent space of equally valid, globally optimal options.

Having addressed these matters, we continue on the next and final chapter of this thesis. In the subsequent chapter, we present our experimental results, discuss our results on the recourse task on several datasets, and our performance compared to other works. We delve into the conclusions of our method and report our limitations.

Chapter 5

Experiments

Having our problem set up and the model appropriately discussed, we now come to the last chapter of the main body of this thesis, where we present our Experimental Setup. In this chapter, we set the preliminaries for our experiments; we briefly analyze the datasets we use for benchmarking, their special preprocessing they need, how we binarized and to what extent we increase each datasets dimensionality. We discuss the challenges each dataset opposes to our work and the competitors works and how we planned to deal with it. Next, we further analyze our evaluation metrics, that still refer to the cost and loss function. In this section the only special detail we will present is the mapping of a cost function that was created to deal with continuous and real numbers, to adapt into a binarized logic. We provide the experimental set up as far as the software and hardware used, are concerned, and present the code significance on the STreeD part of the task.

On the next, and final section we present our experimental results, based on the benchmarking datasets we use and compare our work to other methods. At the same time we contribute more results and visualizations of our method to quantify the impact of the pareto front solutions, their personalization flexibility, and the tradeoff optimal solutions. Finally we conclude with the limitations of our work, and things that could not work out during the process of this thesis.

Contents

5.1 Preliminaries

In this section, we briefly analyze our experimental setup, referring to our software contributions, the hardware equipment, and briefly state the needs of computational power. We then discuss our datasets, and the preprocessing adapted and extended for our SOGAR method under the STreeD solver. Finally, we restate our evaluation criteria via the cost and loss function, and discuss the experimental modifications to be able to map the action space from the binarized feature space to the prebinarized feature values.

5.1.1 Experimental Setup

In this section, we briefly analyze our experimental setup, referring to our software contributions, the hardware equipment, and the computational needs. Our implementation runs on an **Apple Silicon M4** CPU with **16 GB RAM**. The SOGAR optimization task is implemented as a new STreeD-defined separable task and integrated with the STreeD solver back-end. The remaining sub-processes of the formulation, including preprocessing and caching, are implemented in **Python 3.12**, while the STreeD task is compiled in **C++17**.

We adopt the *replicable* configuration of the CET repository[1] for the classifier and use **LightGBM**[30] with the default parameters the CET authors use, to replicate their experiments and ours with compatible versions and parameters, and to ensure comparability across methods. At this point, we emphasize that runtime scales primarily with the *number of instances*, that is, the population size of the affected set, rather than the number of features. This observation is consistent with the use of a fully binarized search space and a cache-based evaluation that produces $O(1)$ leaf-score retrieval per (x, a).

5.1.2 Datasets

We evaluate on four tabular datasets. We run experiments on three UCI datasets that are standard for auditing risk-sensitive decisions: **German Credit**[31], **Bank Marketing** [32], and **Adult Income** [33], and the IBH HR **Employee Attrition**[34]. These datasets are chosen to implement the *auditing* task—global, population-level summaries of recourse, in order to compare with related works, and specifically the CET replicable repository that except of CET, includes the only available public implementation of AReS, and a cluster-wise implementation of actionable recourse for broader comparison of methods.

The first preprocessing of the data adapts the one-hot encoding technique for categorical features, as in the CET repository. This dataset is used for the training of the classifier, the evaluation of costs applied to the original values of instances, and for inferring the edited instances to the trained model. Thus, the binarization we implement has the following extra steps:

- We preserve the already binary, and one-hot encoded categorical features,
- We handle continuous features by binarizing them into appropriate interval bins, utilizing the *quantile binning* of intervals that the STreeD framework is compatible with, and is proven to utilize with higher efficiency. For large real-valued ranges, we use interval aggregation by choosing the optimal representative ranges, and a representative value based on the bin's median to be saved in the context file. For smaller feature distributions that can be binarized into distinct bins for each value, we adapt this approach to exactly match the cost of actions that will be later discussed in the subsequent subsection 5.1.3.
- We control the dimensionality post-binarization, as binning radically can increase the number of binarized features that will be included in the STreeD solver. Thus, we aim to control such increases by at most $\approx 100 - 120\%$ in edge cases. However, such binarization does not seem to affect the resulting solution scores, based on several experiments, and is fully necessary for the correct feature splits and mapping in the binary only values of the STreeD solver, without loss of information by just creating binary features with one threshold over a widely ranged distribution.

Dataset sizes (original space). Table 5.1 reports original feature counts and the number of instances used, before the one-hot encoding of categorical features. The *Adult* dataset is subsampled to half the original instances to allow baselines to complete, while preserving class ratio and the geometric integrity of

[1] https://github.com/kelicht/cet

the feature space. Also the Bank dataset is the smallest variant of the original dataset that represents a random 10%, which the authors chose to distribute as an extra file.

Table 5.1: Original dimensionality and instances used.

Dataset	**# Features (original)**	**# Instances (used)**
Employee Attrition	34	1470
German Credit	20	1000
Bank Marketing	16	4500
Adult Income *	14	16000

Post-binarization. Table 5.2 summarizes the expansion after one-hot and interval binning, as well as the immutability portion of the feature space and action configuration per dataset. The immutability percentage denotes the fraction of features that are flagged as immutable in the context file, and therefore excluded from edits. Column "Actions/inst." denotes the maximum combination of atomic edits that are allowed to exist, thus constraining the feasibility of the sparse action vector, which preserves the minimal change for maximal flip. This constraint, however, was chosen in the same manner as the works we compare with, for compatibility and fair comparison. It can be extended to larger, yet controllable combinations, to preserve sparsity. The limitation of such a choice of expansion of the action space, however, creates a rather computationally challenging space, and the more atomic actions exist, the greater the number of combinations will be. The last column reflects the edit space cardinality per instance, as it reflects the size of the precomputed action set used during DP search.[2].

Table 5.2: Post-binarization dimensionality, Immutability, & Action set cardinality.

Dataset	**#Feat.**	**# Bin. Feat.**	**Δd%**	**Immutable (%)**	**Actions/inst.**	**# actions**
Employee Attrition	45	100	120.0	26.1	$k \leq 3$	68,147
German Credit	41	90	119.5	19.0	$k \leq 3$	62,440
Bank Marketing	35	77	120.0	65.0	$k \leq 3$	20,105
Adult Income *	108	172	59.3	60.2	$k \leq 3$	15,624

In order to provide some extra insights on the preprocessing and preparation of actions, the binning increases the feature space to more than double of the feature dimension used in the dataset that was used in the training of the classifier training and inference. However, this allows us to create better and more logical splits that absorb the maximal information by the binarized dataset that approximately maps the original. The action space is significantly larger than the other works, as we take the DP problem solving to the full extent by implementing an exhaustive search over all possible sub-solutions.

Then we see that the percentage of immutable features varies per dataset. This was personally tailored by us, and some already given suggestions by previous works on the respective datasets. The immutability of features, was chosen based on criteria of equal opportunity and elimination of discrimination, so the main types of features that were set as immutable were, *Age, Gender, Race, Marital Status, Relationship* and *Country*.

The carefully tailored actions consist of edits that depend on the type of feature. Numeric features are mapped such that the change of bin that is chosen in the DP solver, thus the command to move to another range, is translated by a move to reach the median of the target bin. The integer features are handled similarly, but each bin contains the value of the integer, instead of a range. The binary actions are handled as only activating or deactivating the specific feature, and we adapt the exact same way as the categorical features.

The extent of bins is clearly shown in the first three datasets of the table, of Employee Attrition, German Credit, and Bank marketing, as we expanded columns like *income* to $20 - 50$ bins that represent ranges, so

[2]In our experiments, we cap edits per action by $k \leq 3$, as discussed in Section 4.4

that we can provide actions of not too large edits, thus unrealistic income increases. We specifically make the bins have approximately equal distance between every to bins. This is chosen so that the actions we will propose will have a standard move, fixed to this distance, and then multiplied by a chosen maximum allowed number of jumps. Thus, actions that suggest $+500, +1500, +3000$ increase in income are suggested over actions seen on other works that suggest individuals increase to more than $7,000 - 10,000$. These changes provide a more sophisticated approach to creating actions that are applicable to every instance and will likely never lead to overfitting on the needs of a rather small portion of individuals. This approach is designed for the utmost goal of providing feasible actions that would prove their optimality not only on the training set, but on their performance of success on the test process.

5.1.3 Evaluation Metrics

As we previously stated in section 4.3, we evaluate each tree by two leaf-additive objectives, a flip loss that counts misclassified (non-flipped) instances after applying a prescribed action, and a scale-invariant cost metric based on the *Maximum Percentile Shift* (MPS). The loss is the standard indicator used in counterfactual recourse and in CET, chosen for its simplicity and correctness as a flipping test. The cost is MPS, chosen for its ability to compare heterogeneous features on a common percentile scale. We continue to remind the metrics we use for evaluation, by restating them and giving an additional insight on our practical implementation on our code, so that we can map actions and their costs from the binarized feature space to the original feature space.

First, let us redefine the loss function. Given a target label $y^\star = +1$ and an action a applied to an instance $\mathbf{x}$, write $\mathbf{x}_i^{(a)}$ for the instance after applying a to $\mathbf{x}_i$. The flip loss is

$$\mathcal{L}_{\text{flip}}(f; a) = \frac{1}{n} \sum_{i=1}^{n} \delta_{f(\mathbf{x}_i^{(a)}) \neq y^\star}, \qquad \delta_{f(\mathbf{x}_i^{(a)}) \neq y^\star} = \begin{cases} 1, & \text{if } f(\mathbf{x}_i^{(a)}) \neq y^\star, \\ 0, & \text{otherwise.} \end{cases} \tag{5.1.1}$$

The loss aggregates additively over leaves and over the affected population. We adopt the same convention as in CET for evaluation, solely to serve reproducibility and a clean comparison to published baselines.

Next, let Q_j denote the empirical cumulative distribution function (CDF) of feature j, computed on the original dataset after the prediction of the classifier, thus in a concatenated format of the training and test sets. For instance, $x$and action a, define $x' = x + a$ as the edited instance in the *original feature space*. The MPS [23] cost is

$$c_{\text{MPS}}(x, a) \;=\; \max_{j \in \mathcal{A}(x)} \Big[Q_j(x'_j) - Q_j(x_j) \Big], \tag{5.1.2}$$

where the $\mathcal{A}(x)$ denotes the space of actionable features based on the constraints and the nature of the feasibility set. This choice is *scale-invariant* across features, as the same percentile movement has the same cost, irrespective of units, variances, or support. In this thesis, MPS is *implemented exactly as in the authors' public CET repository* to ensure strict reproducibility and one-to-one comparability of scores across works. Any difference in the implementation would make our work ineligible for comparison, or would need the reproduction of the other works to match a function defined in a totally different way.

We do not need to delve into the *feature-type handling in MPS*. Each feature is handled differently based on its type, and in our work, the additional pre-processing of data needs extensive explanation of how we handle and map our costs. In the following featured outline, we briefly yet sufficiently state the way each feature is handled and any differences that occur in our mapping. The CDF component $Q_j$is instantiated in a feature-wise manner:

- **Binary features** $x_j \in \{0, 1\}$. The empirical CDF has a single jump at 1. If an action flips $0 \to 1$, then $Q_j(1) - Q_j(0) = \Pr(X_j \leq 1) - \Pr(X_j \leq 0) = 1 - \Pr(X_j = 0) = \Pr(X_j = 1)$, i.e., the cost equals the empirical frequency of the 1-category. (The reverse $1 \to 0$yields the symmetric decrement.)
- **Categorical features** (one-hot). Each category is a binary indicator after one-hot encoding, so switching categories is treated as switching one indicator $0 \to 1$and the previous one $1 \to 0$on separate coordinates. The shift on each coordinate is computed as for the binary case above. The overall MPS remains the coordinate-wise maximum as in (5.1.2).

- **Discrete integers.** We construct $Q_j$from the CDF over distinct values. An edit that maps $x_j$to x'_jincurs the percentile change $Q_j(x'_j) - Q_j(x_j)$determined by those discrete cumulative masses.
- **Continuous real features.** In this kind of feature, it is more realistic to use the empirical CDF on the feature distribution, as we have a large range of values and a far larger sample that varies. In our pipeline, the solver operates on a binarized representation, but costs are computed in the original space. To map bin decisions back to real features in a stable and reproducible way, we associate with each bin its *training-split median* $\text{med}_b$and evaluate

$$Q_j(x'_j) - Q_j(x_j) \;=\; Q_j(\text{med}_{b^\star}) - Q_j(x_j),$$

  where $b^\star$is the target bin selected by the action on feature j. This median mapping provides a robust representative within each bin and avoids edge-instability near bin boundaries. All medians and bin boundaries are recorded in the context file and kept fixed across train and test sets.

At this point, we need to note that all $Q_j$are computed once on the post-prediction dataset with original data values and serialized in the context. Then, boundary comparisons in dominance tests use a small εtolerance as we stated on Section 4.4 to mitigate floating-point artifacts and tighten the search and computation space and distress the runtime.

Having referred to the above, we need to make another note on the choice and use of the specific cost function. The MPS communicates the *difficulty*, and thus, an approximation of the rarity of moving to a more advantageous region of the population distribution, without feature-specific scaling or ad hoc weights. This is appropriate for our auditing goal, because we compare and aggregate actions over heterogeneous attributes on a uniform, interpretable scale. We remark, however, that some applications may prefer utility-weighted costs (e.g., domain choices trading payroll increases against stock options). Such preferences can be incorporated by replacing $Q_j$with calibrated utilities or by post-hoc reweighting. We deliberately avoid these domain weights in this chapter to preserve fairness and balance of comparison and to maintain strict compatibility with CET's public implementation.

Having discussed all the details of the preliminaries of our work, given the experimental set-up, the pre-processing techniques followed, the implementation of our actions, and the analysis of metrics, and every component of this work, we now move to the experimental results based on the experiment we ran on the previously referred datasets.

5.2 Results

In this section, having set the ground on the theoretical and practical background, and having collected all the needed preliminary knowledge, we present the results of the experiments we ran on the four tabular datasets discussed in section 5.1.2. We present our solutions, their format, the quantity, and their quality based on the Pareto front, the personalizable parameters of scaling the cost function, and the explainable part that is enhanced through our representation that exploits the STreeD framework. Then we continue to present our results compared to related works of the Actionable Recourse Summaries field, and understand what we achieve by this process, where our results show similarity, what our benefits are, and which parts show extensive importance of discussion. We then conclude this section by discussing parts of significant importance that we observed via our experiments, and present our limitations.

5.2.1 Solutions and Evaluation

For our experiments, we run every work using the same global parameters to evaluate the cost and loss metrics, for the global view of each method. We set $\gamma = 0.99$ for an insignificant reduction of the importance of the flip loss metric. The rest of the parameters, as far as the evaluation is concerned, are left untouched. In our method, we do not use the λ parameter of regularization that the CET authors use, because we are given full control over the maximum depth d, minimum leaf size m, and maximum number of internal nodes M. The rest of the parameters are set to default, exactly as set on the replicable implementation of the works we compare with. For the AReS implementation, we were unable to run the experiments on the Adult Income dataset, even though it was sub-sampled, as the combinatorial nature of the problem made it more

than inefficient for the experiments to run on our current hardware. Thus, no results are returned for the AReS method as far as the Adult income is concerned. The methods of CET and Cluster-wise Actionable Recourse originally utilize the CPLEX MILP optimizer to examine the optimal actions search. However, due to the compatibility of the code implementation and some technical issues, we used the Gurobi solver instead, which is equivalent to the former implementation.

Before we compare and comment on results, we should have a more detailed view of our solutions, their nature, the scores evaluation through consistency of solutions, the number of solutions, the Pareto front formulation, and the visualization of experiments on our examined datasets.

Extent and pruning of solution space

To begin with, in our experiments on **SOGAR**, the number of solutions in every run varies, based on the parameters set on the tree structure. The maximum allowed depth d, the maximum allowed number of internal nodes n, the minimum number of a leaf's size m (the minimum allowed number of instances that should be contained in a leaf, for a tree to be considered a valid and feasible solution), the extent of the of the Pareto front bounds and the relaxations ϵ, all play a crucial part on the cardinality of the solution set. A set of parameters that refer to a shallower tree (e.g, $d = 2, n = 3$) and a dataset of a small to moderate size ($1000 - 2000$ instances, such as Attrition and German) returns a set that ranges from $\approx 180 - 220$ non-dominated solutions. By increasing the depth and number of internal nodes parameters, we extend the search space and thus the Pareto front size to range between $\approx 250 - 500$ solutions. Of course, we need to distinguish here that our comments refer to the given number of actions assigned to the respective action sets computed for each dataset. Naturally, the increase in the size of the action set contributes significantly to the extent of the search space by multiplying the possible solutions and candidate leaves that would occur. In our current setting, however, the number of solutions will be discussed under the specifically stated cardinality of every action set $\mathcal{A}_)(x)$ as observed in table 5.2. On datasets of medium size, such as the Bank dataset, a set of solutions for the shallower trees and a significantly lower number of actions, escalates to have at $\approx 250 - 500$, exactly like the case of deeper trees for smaller datasets. Similarly, the largest of the datasets, Adult Income, even though it had the fewest number of actions, returned a significantly greater range of $2000 - 3000$ solutions for both $d = 2, n = 3$ and $d = 3, n = 7$ combinations.

These observed solutions can, however, be filtered on demand via various techniques. The STreeD framework allows the modification of the merging rules, aiming to control the candidate leaves that will later contribute to the construction of sub-tree solutions, up to the maximum allowed depth, something that, if used, restricts the search space of combinations drastically. Secondly, we tried to filter out actions from the cache that had low scores on the prediction objective. This indeed contributes equally to the pruning of the search space, but it is an unorthodox way, and considered to be wrongfully implemented, as the filtering was done under the average loss of actions when examined on the whole population. This was not included in the final implementation, but an approach that would prove a filtering technique would definitely benefit the process. The third and most useful method to provide an indirect pruning of the search space was the use of the ϵ relaxation constraint on the Pareto front dominance conditions of comparison. Larger values of ϵ compare on fewer decimal digits, thus the dominance of solutions was more efficiently computed. Such a constraint is valid in our results, as prompted by the literature on multi-objective optimization problems, and used to lessen the comparisons and the demanding accuracy of floating point solutions.

Nature and consistency of solutions

In the context of the nature of solutions, we refer to the different visual and qualitative results. The extent of interpretability of the tree structures and the explainability based on the actions provided per leaf contribute to the understanding of the examined classifier's behavior. As depicted by figure 5.2.1, the tree structure is concretely explainable using the feature binary splits to guide to the appropriate action assigned on leaves. In the figure, we see a balanced structure of four leaves, each meeting the criterion of the minimum number of instances per leaf that was set to $m = 30$. The structure offers full transparency and consistency through the feature splits and on the actions assigned, which contain the number of instances of the training process and the number of training instances reached on testing. Except for the reference to instances, we extract the most important part of the output, which is the training and testing scores per leaf. We observe that

the average cost generalizes sufficiently well on the test instances, which is equally observed on the train and test loss scores.

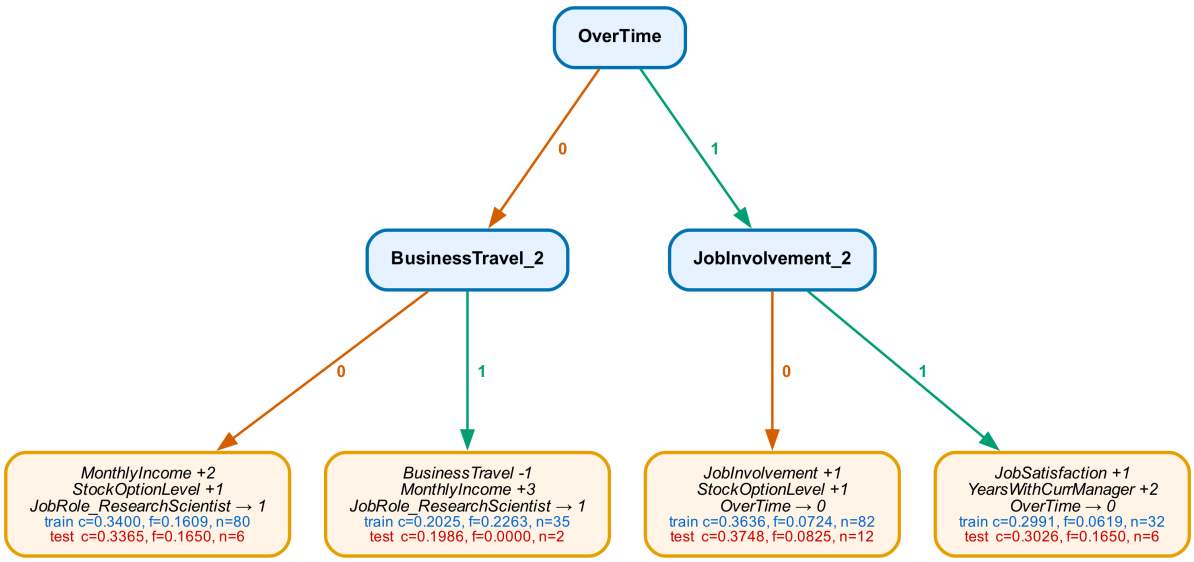

Figure 5.2.1: SOGAR tree of depth $d = 2$, Attrition dataset

The tree represents the output of the solution providing the minimal training invalidity score, for an experiment run on the Attrition dataset, for parameters of depth and maximum number of nodes $d = 2, n = 3$, respectively.

Similarly, we obtain another example of the same dataset, which returns a tree of depth $d = 3$, as seen in figure 5.2.2. In this situation, things differ in the of generalization. This is a logical aftereffect of allowing a tree that is training on a small population to grow deeper. As a result of such a relaxed constraint, even though the training scores are improved, the testing scores are worsened. This observation is, in fact, useful for the user to understand the nature of the solutions because, even though the actions proposed by the deeper structure are more specified on the partitioned population, the generalization is effectively declining, allowing us to calibrate the complexity of the structure provided the scores evaluation. Nevertheless, such small differences in scores do not directly indicate the quality of the actions proposed. On the other hand, the user should criticize the quality by comparing both the complexity of solutions and the rationale of each proposed edit.

The second aspect discussed, based on the solution of 5.2.1 is the rest of the non-dominated solutions that were included on the Pareto front and the consistency of scores as we decline in the ranking based on the training invalidity score. Specifically, on table 5.3, we observe the difference through the ten best-ranked solutions that were returned on the Pareto front. The step-wise difference rate of the training invalidity is by average $+0.28\%$ and the average per-step change is $+0.00125$. This indicates the advantage of consistency on the best-scoring solutions. This provides the user a flexible choice between multiple solutions that are approximately equally ranked, and each offers a different tree structure, which mostly differs in the choice of actions. In other words, a user can choose freely between the best solution that suits their needs, without being severely penalized.

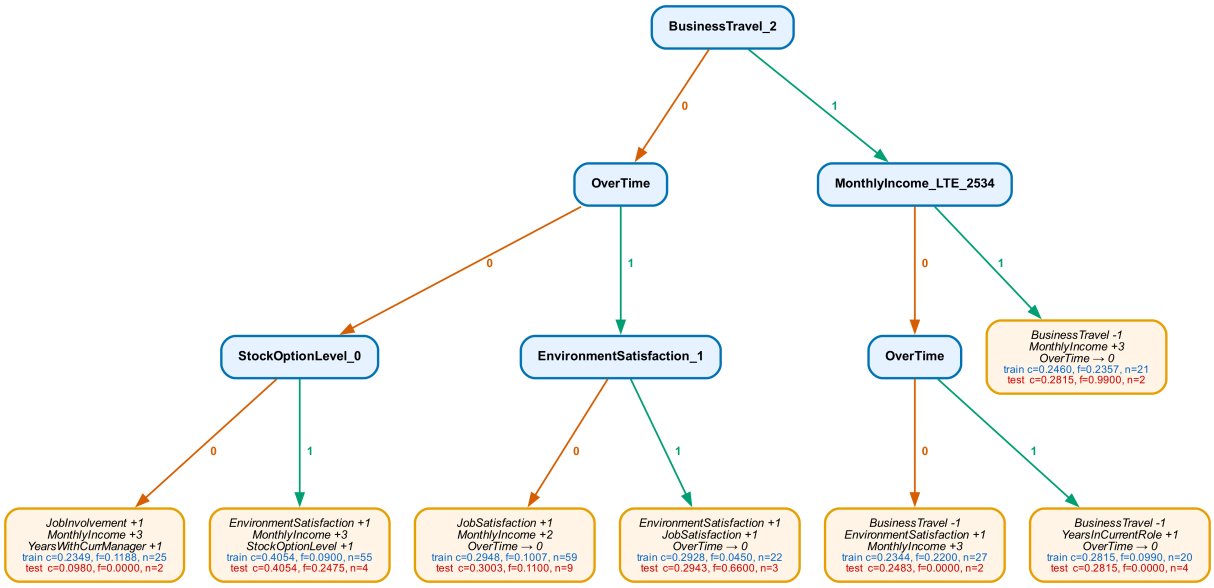

Figure 5.2.2: SOGAR tree of depth $d = 3$, Attrition dataset

The tree represents the output of the solution providing the minimal training invalidity score, for an experiment run on the Attrition dataset, for parameters of depth and maximum number of nodes $d = 3, n = 7$, respectively.

Table 5.3: Top 10 - Solution Metrics Comparison Attrition ($d = 2$)

Solution	**Train**			**Test**		
	Cost	**Loss**	**Inv.**	**Cost**	**Loss**	**Inv.**
Solution 0	0.3217	0.1254	**0.4471**	0.3357	0.1142	0.4500
Solution 1	0.3184	0.1297	**0.4481**	0.3361	0.1142	0.4503
Solution 2	0.3168	0.1340	**0.4508**	0.3380	0.0761	0.4142
Solution 3	0.3310	0.1210	**0.4521**	0.3551	0.1142	0.4693
Solution 5	0.3153	0.1383	**0.4537**	0.3286	0.1142	0.4428
Solution 4	0.3068	0.1470	**0.4538**	0.3127	0.1142	0.4269
Solution 7	0.3121	0.1426	**0.4547**	0.3289	0.1142	0.4431
Solution 6	0.3035	0.1513	**0.4548**	0.3130	0.1142	0.4272
Solution 8	0.2972	0.1600	**0.4571**	0.2974	0.2285	0.5259
Solution 9	0.2972	0.1600	**0.4571**	0.2974	0.2285	0.5259

The table corresponds to the ten (10) best candidate solutions of the Pareto front, ranked based on their training invalidity score. The data correspond to an experiment run on the *Attrition dataset*, using parameters of maximum depth $d = 2$ and maximum number of internal nodes set to $n = 3$. Solution 0 numbers correspond to figure 5.2.1

Additionally, we observe that the same applies for the ten best-ranked solutions for the depth $d = 3$ solution, as the increase rate of invalidity is even smaller, at an approximately $+0.00037$ per-step increase as indicated by table 5.4. Overall, the solutions provided by the Pareto front not only offer a wide variety of solutions that use the cost and loss trade-off differently, but also a significant set of similarly ranked solutions that can be utilized by personal criteria of each user and the setting of each task.

Table 5.4: Top 10 - Solution Metrics Comparison Attrition ($d = 3$)

Solution	Train			Test		
	Cost	Loss	Inv.	Cost	Loss	Inv.
Solution 0	0.3019	0.1211	**0.4229**	0.2919	0.2285	0.5203
Solution 1	0.3075	0.1167	**0.4242**	0.2920	0.2285	0.5204
Solution 2	0.3432	0.0821	**0.4254**	0.3559	0.1142	0.4702
Solution 3	0.3349	0.0908	**0.4257**	0.3484	0.1142	0.4626
Solution 4	0.3349	0.0908	**0.4257**	0.3484	0.1142	0.4626
Solution 5	0.3176	0.1081	**0.4257**	0.3127	0.1523	0.4650
Solution 6	0.3394	0.0865	**0.4259**	0.3547	0.1142	0.4689
Solution 7	0.3310	0.0951	**0.4262**	0.3472	0.1142	0.4614
Solution 8	0.3008	0.1254	**0.4262**	0.2919	0.2285	0.5203
Solution 9	0.3138	0.1124	**0.4262**	0.3000	0.2285	0.5285

The table corresponds to the ten (10) best candidate solutions of the Pareto front, ranked based on their training invalidity score. The data correspond to an experiment run on the *Attrition dataset*, using parameters of maximum depth $d = 3$ and maximum number of internal nodes set to $n = 7$. Solution 0 numbers correspond to figure 5.2.2

Pareto front visualization

Before we delve into the comparison of our scores with other methods, we need to discuss a final aspect of our solutions. The Pareto front formulation can be visualized in Figure 5.2.3, for the experiment run on tree depth $d = 3$. Here, each point depicts a feasible and non-dominated solution. The visualization depicts the expected slope of points to be created, as we can see all the solutions that span the lower levels of cost score, to have an inversely proportional relation to the loss function, and vice versa. This can be logically explained, as the solutions that are denoted to have a smaller cost, thus overall the actions assigned need the least effort compared to ones with higher costs, are likely to mostly fail to flip the label of the majority of the affected population of instances. Similarly, the points of the plot that indicate having greater cost scores are more likely to succeed in flipping the label, as observed on the points of the lower right of the graph.

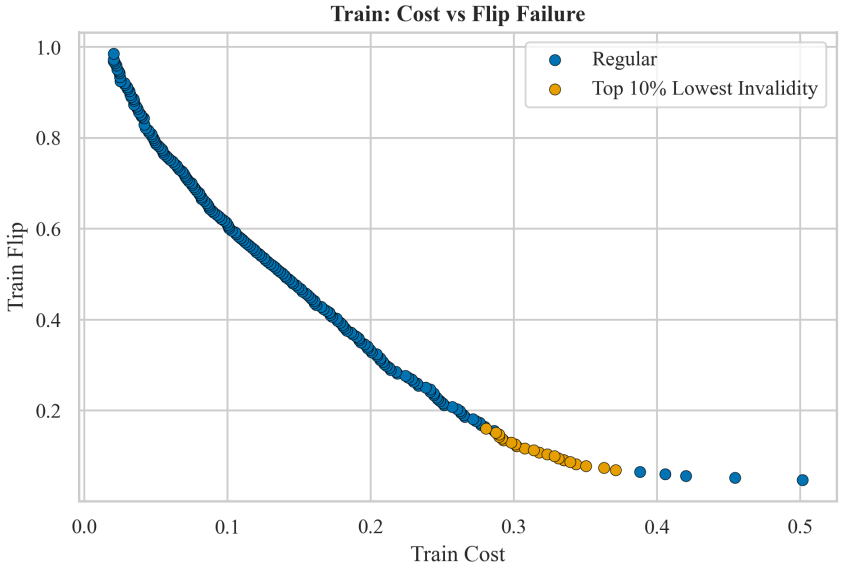

Figure 5.2.3: Pareto Front - Training Solutions ($d = 3$)

The solutions that were denoted earlier as the ten best-ranked solutions of table 5.4, and a handful of solutions

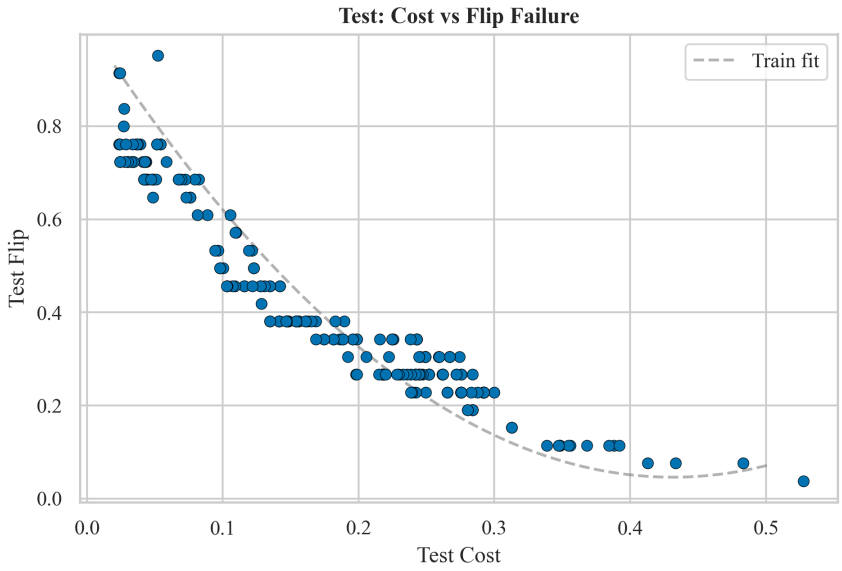

Figure 5.2.4: Test scores generalization ($d = 2$)

that together combine into the best 10% of solutions ranked on training invalidity, are annotated with ocher coloring. The points of this region, as the table suggests, occupy the lower flip loss segment of the graph, such that the loss between solutions differs by minimal amounts, and the cost varies as well between $0.25 - 0.35$. One would suggest that these solutions are visualized to lie in a region where the solutions settle on a specific optimal loss. However, this would be highly inaccurate in such a setting because the loss metrics are scaled and controlled by the γ hyperparameter. As we have previously stated the choice of this scaling factor, calibrates the importance of the effectiveness compared to the cost objective. In our scenario, a slightly reduced loss metric returns solutions that should balance the two objectives, and in different scenarios the region where the highest invalidity scores could dramatically change. The theoretical optimum of solutions derives from the minimal result of the sum of the two objectives (having the loss scaled), which is not visually guaranteed in a specific region. We should acknowledge, however, that additional to this statement, in such optimization cases where the multi-objective optimization is examined, we could examine alternate ways of having a scoring evaluation, such as minimizing the distance of the Pareto solution point from the origin, as for our task the ultimate goal is to minimize the objectives simultaneously. This shows another form of the resilience of choosing which solutions matter the most, based on the context of the task and the users interest.

Nevertheless, the observations of the Pareto front formulation have an additional aspect of interest that provides more robustness to the solution scores. As depicted in Figure 5.2.4, we provide a best fit curve of the training points of the Pareto front solutions, and the corresponding points of the test solutions' scores. The points here do not resemble a Pareto front, as they are the outcome of inference based on the trained tree structure of each non-dominated solution. However, as expected, the generalization we claimed in the previous paragraphs has another point of support, via the behavior of the depicted points that concentrate near the Pareto front curve. The slight variability of the points is interpreted as the small size of the test set (≈ 20 cases), which introduces discretization and random fluctuations in the measured rates. This suggests finite-sample uncertainty, rather than any systematic deviation from the training trade-off.

Having covered the topic of the solutions that **SOGAR** returns, their efficiency and consistency based on the sole ranking of the solutions via the training invalidity score and the Pareto front visualization and interpretation, and having shown the qualitative advantage of such interpretable structures, the next step is to understand how our work performed compared to related works that solve the same optimization problem.

5.2.2 Results Comparison

In this section, our objective is to understand how our method compares to other works, based on the training invalidity score. However, this metric alone should not be sufficient for a global understanding differences of our work compared to the previous works. For that reason, our discussion on results will contain the cost and loss objectives separately, compared to other works, for training and testing evaluation. The last part of the comparison includes the runtime comparison of each method, and in what manner is this comparison correlates with the score efficiency of our method compared to the related ones.

Before we discuss the performance of each method, we need to note that the experiments on the CET method for the Adult dataset were run on $T = 1000$, compared to the other datasets that were run on the default parameter that was indicated in the authors' code of $T = 3000$, because of our insufficient computational resources and the demanding runtime the experiment needed. The results match the mutability percentages of table 5.2.

To begin with, the results of Table 5.5 provide an analytical presentation of the performance of each method, when run under a 10-fold cross-validation, on the four datasets, using the previously noted default parameters on the Cluster-wise Actionable recourse, the AReS, and CET methods as were prompted on the original reproducible implementation by [1]. Our method was run on two main sets of parameters of depth and number of internal nodes, that were **(i)** $d = 2, n = 3$ and **(ii)** $d = 3, n = 7$ and an appropriate empirical tuning of the size of leaves, depending on the depth allowed and the size of the dataset. In further detail, the leaf sizes for the small and moderate datasets were set to a minimum of $m \approx 20 - 30$, and as we scaled the instance space on the Bank and Adult datasets, we set a minimum of $m \approx 100 - 200$ and $m = 300$, respectively. The results referenced in the cross-validation results table are representations only of the depth $d = 3$ for every dataset.

Our initial observation is that the **SOGAR** method has a significant improvement over all the compared methods, on the training invalidity score, and consequently on the test invalidity score. The mean difference is on the scale of $\approx 40\%$ on both training and test invalidity, something that derives from the significantly lower loss objective. The crucial factor of this minimal loss is interpreted via three reasons that we suggest, based on our understanding and knowledge of the related implementations and the benefits of the Optimal Decision Tree structures. We will analyze our justification in a declining order of importance.

Table 5.5: Results of 10-fold cross-validation

		Train			**Test**		
Dataset	**Method**	**Cost**	**Loss**	**Inv.**	**Cost**	**Loss**	**Inv.**
Attrition	Clustering	0.068 ± 0.05	0.940 ± 0.04	0.999 ± 0.01	0.0084 ± 0.11	0.918 ± 0.13	0.992 ± 0.03
	AReS	0.442 ± 0.05	0.439 ± 0.09	0.877 ± 0.06	0.445 ± 0.07	0.285 ± 0.09	0.728 ± 0.08
	CET	0.294 ± 0.12	0.477 ± 0.15	0.776 ± 0.07	0.305 ± 0.12	0.45± 0.21	0.751 ± 0.14
	SOGAR	0.302 ± 0.032	0.135 ± 0.03	↓ **0.437 ± 0.035**	0.295 ± 0.021	0.1297 ± 0.060	↓ **0.425 ± 0.066**
German	Clustering	0.042 ± 0.02	0.916 ± 0.04	0.949 ± 0.03	0.046 ± 0.02	0.925 ± 0.05	0.962 ± 0.03
	AReS	0.452 ± 0.09	0.232 ± 0.05	0.683 ± 0.11	0.467 ± 0.12	0.265 ± 0.08	0.732 ± 0.14
	CET	0.107 ± 0.02	0.269 ± 0.01	0.374 ± 0.07	0.108 ± 0.02	0.226 ± 0.11	0.332 ± 0.11
	SOGAR	0.109 ± 0.007	0.086 ± 0.022	↓ **0.195 ± 0.019**	0.109 ± 0.013	0.111 ± 0.058	↓ **0.220 ± 0.059**
Bank	Clustering	0.010 ± 0.02	0.993 ± 0.01	0.993 ± 0.012	0.607 ± 0.010	0.994 ± 0.011	0.994 ± 0.0
	AReS	0.361 ± 0.02	0.799 ± 0.08	1.152 ± 0.09	0.363 ± 0.03	0.798 ± 0.08	1.152 ± 0.08
	CET	0.035 ± 0.003	0.018 ± 0.01	0.053 ± 0.01	0.035 ± 0.01	0.019 ± 0.01	0.054 ± 0.01
	SOGAR	0.029 ± 0.0004	0.007 ± 0.006	↓ **0.037 ± 0.006**	0.029 ± 0.0006	0.011 ± 0.007	↓ **0.039 ± 0.007**
Adult	Clustering	0.95 ± 0.0	0.107±0.01	1.056± 0.01	0.948± 0.04	0.094±0.06	1.041±0.08
	AReS	—	—	—	—	—	—
	CET	0.935 ± 0.04	0.110±0.012	1.045± 0.02	0.94± 0.04	0.099±0.05	1.039±0.08
	SOGAR	0.166 ± 0.005	0.448 ± 0.007	↓ **0.626 ± 0.002**	0.172 ± 0.005	0.435 ± 0.002	↓ **0.607 ± 0.002**

The 10-fold cross-validation on the LightGBM classifier [30].

First, we have previously stated numerous times that the dynamic programming formulation of optimal decision trees, which explores the entire feasible sub-tree space through systematic enumeration and Pareto dominance pruning, is the greatest asset that this formulation offers. To frame this under the results of the compared works, we comprehend that all the possible different options of equally optimal leaves and structures

that were examined by the STreeD solver were indeed able to retrieve globally coordinated combinations of leaves that minimize loss while maintaining equivalent average cost. This indicates the use of the optimal splits found per run, and justifies the improvement of the loss function related to previous work, which often relies on locally optimized or heuristic tree construction methods. The CET algorithm, indeed, optimizes actions locally within each leaf, based on its formulation. However, this local optimization does not guarantee global coordination across the entire tree structure, but rather depends on the nature of the leaves and the current proposed action rules.

The second argument for such lower results is the evaluation of leaf nodes. Both CET and SOGAR solve an optimization problem on each leaf node. However, the CET formulation, utilizes the mixed-integer linear optimization (MILO), which by linear constraints computes the optimal actions. These constraints restrict the feasible action space to satisfy flip and feature-weight conditions defined by the classifier, effectively narrowing the exploration region. In contrast, SOGAR evaluates all feasible cached actions, for every candidate leaf that the algorithm examines. This connects directly to our previous argument and extends the fact that the global search, might be more computationally demanding, yet provides a rather hands-on evaluation of possible outcomes.

This leads us to the third point of interest, which is not a proven verdict but rather an observation of the different handling of actions through the methods. In our method, action rules are initiated exactly on the same mutability and directionality criteria as the rest of the works. However, we create the actions independently for every feature and then create all the possible combinations that occur. This leads us to a large action space, which, in contrast to the CET implementation, evaluates more combinations of actions, as the CET approximation group-wise local counterfactual movements derived from linear surrogate models (AR-LIME) [120]. This approach, plainly explained returns actions that propose local "movements" such that the instance will move to a positively classified region. For this reason our hard-coded approach, commands each instance to make specific predefined movements that do not lie on its personal feature values. By contrast, our cache-based approach generates deterministic, predefined edits under mutability constraints, producing a broader yet computationally heavier exploration of feasible actions.

Table 5.6: Average computational time/s

Dataset	**Clustering**	**AReS**	**CET**	**SOGAR**
Attrition	5.57 ± 0.268	307.36 ± 22.67	794.16 ± 34.10	**708.64 ± 10.2**
German	6.61 ± 0.313	572.0 ± 101.51	550.55 ± 21.72	**668.243 ± 34.1**
Bank*	50.72 ± 3.77	855.0 ± 107.41	2061.82 ± 320.48	**3607.5 ± 32.6**
Adult**	271.95 ± 9.19	—	15236.0 ± 1193.4	**8930.88 ± 100.36**

The table shows the per-run core. The bold text values denote our method's time needed to reach the optimum.

To complete the discussion of our results, we need to comment on the case of the costs evaluated by the methods, and the edge case of experiments on the Adult dataset. Starting with the latter, we need to comment on the rather different behavior of the methods as far as the results are returned. Our aim was to have a sample of a larger dataset, to understand how the methods would perform, and what differences would occur on scores. However, it is important to note here that the dataset, was highly constraint by the mutability of features due to the computational complexity that would occur, give our hardware restrictions. These restrictions provide an explanation of how the Cluster-wise and CET formulations were mostly based on solutions that had the least uncertainty of flipping, which was traded for high effort actions. We should also acknowledge the part that CET implementation, was only run for $T = 1000$ iterations, which is definitely a reason of not providing lower cost results. On our implementation the scoring was seen to be better balanced, and our returned results, as we return higher but viable loss results for a much smaller effort, thus returning a lower total invalidity score. This final dataset, however, should be more thoroughly examined to accurately comprehend such model behaviors.

The comparison of the action cost, is a much different topic. The clustering method, is in general the optimal action finder for low effort actions, yet not effective enough to flip the label. The rest of methods on the other hand, return in general an approximately equivalent average cost per dataset, that indicated that the optimal actions are correctly distinguished by the methods, and thus the significant difference between results is the

correct way to partition the leaves such that the actions have the optimal impact on the population. In our method, we need to comment that the average costs might be slightly worse, for example on the Attrition and the German datasets. We justify this based on the way we formulate the actions of our dataset, as the edits on continuous features are hard-coded, as explained in Sections 5.1.2-5.1.3, to reach specific target points of the feature distribution. This was a necessary measure taken, to logically and robustly encode the actions between binarized feature space of the DP solver, and the original feature space that the classifier encodes the feature values.

Finally, before we delve on the last section of this chapter, let us report the final computational time comparison that is essential for the results comprehension of performance. In table 5.6, we provide a reference to the previously compared results of scores. Our first remark, is that the clustering method is by a great margin the fastest to return the actionable resource summary across all methods, and that is justified by the heuristic nature of the method. On the rest of the methods, we observe an equivalent computational time needed on the small to moderate datasets of Employee Attrition and German Credit, with indication that our method is slower compared to the other methods, something that is more thoroughly proven on the Bank dataset where we scale on moderate to medium dataset sizes, an our method needs a handful of an hour computational time. This was an expected behavior based on the expansion of search space to all possible globally optimal solutions. However, these depicted results do not worry as for the SOGAR models performance, as the implementation was solely using serialized processes, and no multi-threading technique was implemented at the time of these experiments whatsoever. By that we state that there still is room for improvement in performance. The last dataset, however, provided a different picture between the computational time needed for the SOGAR and the CET methods[3], that on average have the best performance on invalidity scores. Both methods on average need an order of magnitude larger computational time, for a large dataset as Adult income. These results, are a clear indication that the combinatorial problem of finding such optimal actions, is drastically under-performing as we scale.

5.2.3 Further Discussion and Limitations

As a last part of the discussion, we need to briefly address the key limitations of our work and possible improvements that would return more efficient results. First, although the performance of scores signifies the improvement and a different way to optimally partition the feature space, we acknowledge the drawback of time-consuming and sequential computations. As we stated on the previous Section 5.2.2, the approach utilized a CPU-only hardware approach that was implemented using serialized, single-core processes. For this case, we attempted the formulation of a multi-threaded cache construction and parallel subproblem evaluation for our candidate leaves. However, on the timeline of this thesis, we had no stable version of such an asset, thus our discussion of results only included the single-threaded approach. This limitation originates from the dynamic programming recursion, which depends on previously solved sub-trees and therefore restricts straightforward parallelization.

The second, highly restrictive aspect of this work, was the need for binarization of the feature space. This, feature of the binarized dataset is what makes the DP formulation of STreeD perform and scale, on such a demanding combinatorial task. Nevertheless, the choice of correctly split features into range bins, was a rather difficult and uncertain process. The small number of such bins for continuous features would return results that would be unlikely to be trustworthy. The actions were restricted to large leaps between smaller ranges, and thus the performance would be highly affected, either by choosing sub-optimal features to edit or by selecting optimal actionable features that yield infeasible scores. On the other hand, the larger discretization of continuous features, would be a burden for both the DP formulation that would deal with a large dimension of feature space, and for the preservation of information structure, risking loss of discriminative information. For these reasons we needed to carefully calibrate the number of bins needed so that we would not have to deal with unrealistic results. This binarization trade-off is consistent with other dynamic programming ODT formulations, where binary tests are necessary to preserve separability and enable valid Bellman recursions.

For such continuous feature handling by optimal decision trees, a recent work by the authors of STreeD, the *ConTree* [121], would supposedly deal with our previously referred issue, however, the published implemen-

[3]We again note that the AReS implementation was a technical issue for our work, based on the restrictive time of this diploma thesis, and on various parameter setups, we were unable to retrieve results and thus be able to compare the computational time needed.

tation only supports single-objective optimization tasks, thus a greater amount of time would be needed to implement such a feature, that would totally move us far from the problem of this thesis. Such an implementation would assist our work, to have a more direct comparison to other works, without the need for tailored feature mapping. Given these limitations and the potential additions of our implementation, we aim to have a globally efficient method to solve the problem of Actionable Recourse.

Chapter 6

Conclusion

Actionable Recourse is, indeed, one of the most useful manners to tackle the opacity of black-box models that provide state-of-the-art performance on prediction tasks. It specifically aids general users, practitioners, and researchers to comprehend not only what the model has decided, but also which features and regions of the feature space are viable to obtain the desired prediction.

In our work, we carefully examined formulations that provide solutions to the Actionable Recourse problem. We understood their mechanisms, and were highly influenced and inspired to re-define the formulation of tree structures, that CET introduces, via the Optimal Decision Tree structures. By analytically reviewing decision-tree algorithms and their interpretability benefits, we stated their use when the task refers to interpretable models that can be highly valuable assistants on Explainability Tasks. This led us to thoroughly comprehend the difference of Optimal Decision Trees, and the global optimality factor that they offer, which makes them highly advantageous on tasks that aim to strictly optimize objectives and explore every possible combination, under the concept of separability. This led us to the choice of the Dynamic Programming approach and the highly reproducible and flexible framework of STreeD, that allowed us to adapt the Actionable Recourse task as a bi-objective optimization problem.

This formulation of ours, proved to be a rather promising method, that preserves the structural transparency that previous works had established. The SOGAR method provides a principled foundation that systematically constructs the Pareto front of non-dominated, globally optimal tree solutions. In this manner, we have demonstrated that our solutions are empirically proven, via experiments on several tabular datasets, to increase the effectiveness of candidate actions, by providing unexplored splits that are proven to be optimal under the DP framework. Also, we have shown that the extent of the action set and the evaluation of all actions in the feasible action cache on candidate leaves, even though is time-consuming, remains comparable to related work in our setting and identifies globally coordinated solutions from the tree level down to optimal leaf assignments.

Based on our experiments, we analyzed the Pareto front of non-dominated solutions, and observed a significant increase in successful actions per solution via the lowered flip loss metric. Treating the task as a bi-objective formulation, allowed us to retrieve multiple tree solutions per run. This enables us to personally be able to choose which of these near-tied optimal solutions match our task the best, using the invalidity score as a tie-breaker for selection and comparison to other works. However, we clarify that this manner of comparison was solely our thesis objective, and that the quality of solutions is a personal choice of every user's context and needs of a specific task and domain.

By this, we conclude that the Actionable Recourse Summaries using the Optimal Tree Structure, return a transparent, globally optimized set of recourse policies applying on every instance of an examined affected population, reduces loss and invalidity at a comparable optimal cost, while remaining interpretable and selectable by stakeholders.

Future Work

Given the completion of this thesis, we consider several directions of extending this work, based on the limitations we want to conquer. The first extension, and the most straightforward to implement, is the multi-threaded implementation of the cache prediction and cost computation. In our current setup, the cache creation, which precomputes the action costs and the predicted labels of instances under all feasible actions, is fully serialized. Introducing a parallel process for computing these predictions would substantially reduce the total computational time needed for the cache generation. This is one of the most time-demanding parts of the pipeline, especially when scaling the dataset size, and parallelizing it would allow us to introduce larger action spaces per run and examine larger datasets in a realistic and feasible runtime.

In the same direction, the multi-threaded evaluation of leaf computations inside the STreeD framework would be another important step forward. This, indeed, is technically more challenging, because the dynamic programming recursion depends on previously solved subproblems, it is still possible to introduce controlled parallel execution across independent sub-trees or dominance comparisons. Implementing this would again have a strong effect on runtime, since candidate leaves and the set of actions could be handled concurrently.

Another key direction of further exploration of the problem, is the implementation of a dynamic programming formulation of Optimal Decision Trees that can handle continuous features directly, without requiring full binarization of the dataset. This would allow a more flexible representation of the feature space and make the framework suitable for continuous data, avoiding the possible loss of information that comes from coarse discretization. At the same time, such a formulation should retain the ability to handle the bi-objective optimization task, preserving the balance between cost and flip loss that we introduced in this work.

We should also extend the experimental analysis, by evaluating not only one different dataset, and different rates of immutability of features, but a wider range of state-of-the-art predictive classifiers, such as neural network models or other ensemble-based classifiers, that provide high performance traded for interoperability and explainability of the logical process of internal decisions that produce certain outcomes. Based on the related works, we understood that each model has a diverse way of weighing the importance of features in order to partition the feature space and set decision boundaries. In return, this would clarify at a point, the different behaviors between models and distinguish which of those can be more stable on their decisions and more difficult to change their verdict based on the same actions applied. In such a manner, the summaries retrieved by each model's evaluations would provide a comparable aspect to understand which of the returned solutions hold a more robust, logical and efficient place among all the candidate solutions.

Finally, this thesis sets the ground of the evaluation of the Actionable recourse, under more constraints in the form of additional optimization objectives. Specifically, an extra-mile goals is to apply fairness constraints on the recourse summaries, such that given one or multiple protected features and groups, we would be able to obtain summaries that optimally propose actions that provide low cost and high efficiency actions, in an equally distributed manner across the subpopulations. This at first glance is a highly complex task, that would need further optimization of the implementation frameworks, however, it would be high-impact research, that would examine the biases proposed by the recourse summaries, that of course are directly inherited by the biases of the trained model. It would also aim to provide the outcome of the recourse task, under realistically fair settings, eliminating discrimination across groups.

All in all, while the real hurdles remain, for scalability and continuous feature handling, this thesis advances actionable recourse by coupling optimal decision trees with a principled Pareto-based framework, delivering transparent summaries of feasible and effective actions. With sustained work on parallelization, richer action spaces, continuous optimization, and a broader classifier stress test, SOGAR can mature into a reliable backbone for prescriptive and responsible AI. Continued refinement along these lines would make optimal, auditable recourse a practical default rather than an academic ideal.

Chapter 7

Bibliography

[1] Kanamori, K. et al. "Counterfactual Explanation Trees: Transparent and Consistent Actionable Recourse with Decision Trees". en. In: ().

[2] Linden, J. van der, Weerdt, M. de, and Demirović, E. "Necessary and sufficient conditions for optimal decision trees using dynamic programming". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 9173–9212.

[3] Breiman, L. et al. *Classification and regression trees*. Chapman and Hall/CRC, 2017.

[4] Quinlan, J. R. "Induction of decision trees". In: *Machine learning* (1986), pp. 81–106.

[5] Bertsimas, D. and Dunn, J. "Optimal classification trees". en. In: *Machine Learning* 106.7 (2017), pp. 1039–1082. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/s10994-017-5633-9.

[6] Bertsimas, D. and Stellato, B. "Online Mixed-Integer Optimization in Milliseconds". en. In: arXiv:1907.02206 (Mar. 2021). arXiv:1907.02206 [cs, math]. URL:

[7] Hyafil, L. and Rivest, R. L. "Constructing optimal binary decision trees is NP-complete". In: *Information Processing Letters* 5.1 (1976), pp. 15–17.

[8] Verwer, S. and Zhang, Y. "Learning optimal classification trees using a binary linear program formulation". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 1625–1632.

[9] Aghaei, S., Gómez, A., and Vayanos, P. "Strong optimal classification trees". In: *arXiv preprint arXiv:2103.15965* (2021).

[10] Narodytska, N. et al. "Learning optimal decision trees with SAT". In: *International Joint Conference on Artificial Intelligence 2018*. Association for the Advancement of Artificial Intelligence (AAAI). 2018, pp. 1362–1368.

[11] Bessiere, C., Hebrard, E., and O'Sullivan, B. "Minimising decision tree size as combinatorial optimisation". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2009, pp. 173–187.

[12] Nijssen, S. and Fromont, E. "Mining optimal decision trees from itemset lattices". In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2007, pp. 530–539.

[13] Aglin, G., Nijssen, S., and Schaus, P. "Learning optimal decision trees using caching branch-and-bound search". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 04. 2020, pp. 3146–3153.

[14] Demirović, E. et al. "Murtree: Optimal decision trees via dynamic programming and search". In: *Journal of Machine Learning Research* 23.26 (2022), pp. 1–47.

[15] Chaouki, A., Read, J., and Bifet, A. "Branches: A Fast Dynamic Programming and Branch & Bound Algorithm for Optimal Decision Trees". en. In: arXiv:2406.02175 (2024). arXiv:2406.02175 [cs]. URL:

[16] Ribeiro, M. T., Singh, S., and Guestrin, C. "" Why should i trust you?" Explaining the predictions of any classifier". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[17] Lundberg, S. and Lee, S.-I. "A Unified Approach to Interpreting Model Predictions". In: arXiv:1705.07874 (Nov. 2017). arXiv:1705.07874 [cs]. DOI: 10.48550/arXiv.1705.07874. URL:

[18] Zhang, Y. and Chen, X. *Explainable Recommendation: A Survey and New Perspectives*. 2020.
[19] Pearl, J. et al. "Models, reasoning and inference". In: *Cambridge, UK: CambridgeUniversityPress* 19.2 (2000), p. 3.
[20] Pearl, J. "The algorithmization of counterfactuals". en. In: *Annals of Mathematics and Artificial Intelligence* 61.1 (Jan. 2011), pp. 29–39. ISSN: 1012-2443, 1573-7470. DOI: 10.1007/s10472-011-9247-9.
[21] Molnar, C. *Interpretable machine learning*. Lulu. com, 2020.
[22] Karimi, A.-H. et al. "Model-Agnostic Counterfactual Explanations for Consequential Decisions". en. In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 895–905. URL:
[23] Ustun, B., Spangher, A., and Liu, Y. "Actionable Recourse in Linear Classification". In: *Proceedings of the Conference on Fairness, Accountability, and Transparency*. FAT* '19. New York, NY, USA: Association for Computing Machinery, Jan. 2019, pp. 10–19. ISBN: 978-1-4503-6125-5. DOI: 10.1145/3287560.3287566. URL:
[24] Kanamori, K. et al. "DACE: Distribution-Aware Counterfactual Explanation by Mixed-Integer Linear Optimization". en. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. Yokohama, Japan: International Joint Conferences on Artificial Intelligence Organization, 2020, pp. 2855–2862. ISBN: 978-0-9992411-6-5. DOI: 10.24963/ijcai.2020/395. URL:
[25] Karimi, A.-H., Schölkopf, B., and Valera, I. "Algorithmic Recourse: from Counterfactual Explanations to Interventions". In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '21. New York, NY, USA: Association for Computing Machinery, Mar. 2021, pp. 353–362. ISBN: 978-1-4503-8309-7. DOI: 10.1145/3442188.3445899. URL:
[26] Wachter, S., Mittelstadt, B., and Russell, C. *Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR*. 2018. arXiv: 1711.00399 [cs.AI]. URL:
[27] Poyiadzi, R. et al. "FACE: feasible and actionable counterfactual explanations". In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. 2020, pp. 344–350.
[28] Rawal, K. and Lakkaraju, H. "Beyond individualized recourse: Interpretable and interactive summaries of actionable recourses". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 12187–12198.
[29] Ehrgott, M. *Multicriteria Optimization*. Vol. 491. Lecture Notes in Economics and Mathematical Systems. Berlin, Heidelberg: Springer, 2000. ISBN: 978-3-540-67869-4. DOI: 10.1007/978-3-662-22199-0. URL:
[30] Ke, G. et al. "Lightgbm: A highly efficient gradient boosting decision tree". In: *Advances in neural information processing systems* 30 (2017).
[31] Hofmann, H. *Statlog - German Credit Data*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5NC77. 1994.
[32] Moro S., R. P. and P., C. *Bank Marketing*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5K306. 2014.
[33] Becker, B. and Kohavi, R. *Adult*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20. 1996.
[34] Subhash, P. *IBM HR Analytics Employee Attrition & Performance Dataset*. Accessed: 2025-11-04. 2017.
[35] Mitchell, T. M. "Does machine learning really work?" In: *AI magazine* 18.3 (1997), pp. 11–11.
[36] Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT press, 2016.
[37] Bishop, C. M. and Nasrabadi, N. M. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.
[38] Brown, T. B. et al. "Language Models are Few-Shot Learners". In: arXiv:2005.14165 (2020). arXiv:2005.14165 [cs]. DOI: 10.48550/arXiv.2005.14165. URL:
[39] Morgan, J. N. and Sonquist, J. A. "Problems in the analysis of survey data, and a proposal". In: *Journal of the American statistical association* 58.302 (1963), pp. 415–434.
[40] Kass, G. V. "An exploratory technique for investigating large quantities of categorical data". In: *Journal of the royal statistical society: series c (Applied statistics)* 29.2 (1980), pp. 119–127.
[41] Quinlan, J. R. *C4. 5: programs for machine learning*. Elsevier, 2014.
[42] Ghattas, B., Bouguila, D., and Bennani, Y. "Clustering nominal data using unsupervised binary decision trees". In: *Pattern Recognition* (2017).

[43] Ohl, L. et al. *Kernel KMeans Clustering Splits for End-to-End Unsupervised Decision Trees*. 2024. arXiv: 2402.12232.
[44] Vapnik, V. "Statistical Learning Theory now plays a more active role: after the general analysis of learning processes, the research in the area of synthesis of optimal algorithms was started. These studies, however, do not belong to history yet. They are a subject of today's research activities." PhD thesis.
[45] Hastie, T., Tibshirani, R., Friedman, J., et al. *The elements of statistical learning*. 2009.
[46] Bradford, J. P. et al. "Pruning decision trees with misclassification costs". In: *European Conference on Machine Learning*. Springer. 1998, pp. 131–136.
[47] Lin, J. et al. "Generalized and Scalable Optimal Sparse Decision Trees". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by H. D. III and A. Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 6150–6160. URL:
[48] Breiman, L. "Bagging predictors". In: *Machine learning* 24.2 (1996), pp. 123–140.
[49] Breiman, L. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.
[50] Freund, Y. and Schapire, R. E. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.
[51] Friedman, J. H. "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics* (2001), pp. 1189–1232.
[52] Chen, T. and Guestrin, C. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.
[53] Murthy, S. K., Kasif, S., and Salzberg, S. "A system for induction of oblique decision trees". In: *Journal of artificial intelligence research* 2 (1994), pp. 1–32.
[54] Kontschieder, P. et al. "Deep neural decision forests". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1467–1475.
[55] Land, A. H. and Doig, A. G. "An automatic method for solving discrete programming problems". In: *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Springer, 2009, pp. 105–132.
[56] Wolsey, L. A. and Nemhauser, G. L. *Integer and combinatorial optimization*. John Wiley & Sons, 1999.
[57] Lin, B. and Tang, B. "Optimal Decision Tree - MIP Formulations, Solution Methods, and Stability". en. In: *Integer Programming* ().
[58] Sipser, M. "Introduction to the Theory of Computation". In: *ACM Sigact News* 27.1 (1996), pp. 27–29.
[59] Russell, S. J. and Norvig, P. *Artificial intelligence: A modern approach;[the intelligent agent book]*. Prentice hall, 1995.
[60] Dowling, W. F. and Gallier, J. H. "Linear-time algorithms for testing the satisfiability of propositional Horn formulae". In: *The Journal of Logic Programming* 1.3 (1984), pp. 267–284.
[61] Aghaei, S., Azizi, M. J., and Vayanos, P. "Learning Optimal and Fair Decision Trees for Non-Discriminative Decision-Making". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (2019), pp. 1418–1426. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v33i01.33011418.
[62] Miettinen, K. *Nonlinear Multiobjective Optimization*. Vol. 12. International Series in Operations Research & Management Science. Boston, MA: Springer US, 1998. ISBN: 978-1-4613-7544-9. DOI: 10.1007/978-1-4615-5563-6. URL:
[63] Becerra, R. L. and Coello, C. A. C. "Solving Hard Multiobjective Optimization Problems Using ε-Constraint with Cultured Differential Evolution". en. In: *Parallel Problem Solving from Nature - PPSN IX*. Ed. by T. P. Runarsson et al. Berlin, Heidelberg: Springer, 2006, pp. 543–552. ISBN: 978-3-540-38991-0. DOI: 10.1007/11844297_55.
[64] Laumanns, M., Thiele, L., and Zitzler, E. "Running time analysis of multiobjective evolutionary algorithms on pseudo-Boolean functions". In: *IEEE Transactions on Evolutionary Computation* 8.2 (2004), pp. 170–182. DOI: 10.1109/TEVC.2004.823470.
[65] Demirović, E. and Stuckey, P. J. "Optimal Decision Trees for Nonlinear Metrics". In: arXiv:2009.06921 (Oct. 2021). arXiv:2009.06921 [cs]. DOI: 10.48550/arXiv.2009.06921. URL:
[66] Lipton, Z. C. "The Mythos of Model Interpretability". en. In: arXiv:1606.03490 (Mar. 2017). arXiv:1606.03490 [cs, stat]. URL:
[67] Kim, B., Khanna, R., and Koyejo, O. O. "Examples are not enough, learn to criticize! criticism for interpretability". In: *Advances in neural information processing systems* 29 (2016).

[68] Miller, T. "Explanation in artificial intelligence: Insights from the social sciences". In: *Artificial intelligence* 267 (2019), pp. 1–38.
[69] Costa, V. G. and Pedreira, C. E. "Recent advances in decision trees: an updated survey". In: *Artificial Intelligence Review* 56.5 (2023), pp. 4765–4800.
[70] Brodley, C. E. and Utgoff, P. E. "Multivariate decision trees". In: *Machine learning* 19.1 (1995), pp. 45–77.
[71] Loh, W.-Y. and Zhou, P. "The GUIDE approach to subgroup identification". In: *Design and analysis of subgroups with biopharmaceutical applications*. Springer, 2020, pp. 147–165.
[72] Clemmensen, L. et al. "Sparse discriminant analysis". In: *Technometrics* 53.4 (2011), pp. 406–413.
[73] Wang, X. et al. "Fuzzy rule based decision trees". In: *Pattern Recognition* 48.1 (Jan. 2015), pp. 50–59. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2014.08.001.
[74] "Inducing non-orthogonal and non-linear decision boundaries in decision trees via interactive basis functions". In: 122 (). ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2018.12.041.
[75] Yuan, Y. and Shaw, M. J. "Induction of fuzzy decision trees". In: *Fuzzy Sets and systems* 69.2 (1995), pp. 125–139.
[76] Sosnowski, Z. A. and Gadomer, Ł. "Fuzzy trees and forests—Review". en. In: *WIREs Data Mining and Knowledge Discovery* 9.6 (2019), e1316. ISSN: 1942-4795. DOI: 10.1002/widm.1316.
[77] Frosst, N. and Hinton, G. "Distilling a neural network into a soft decision tree". In: *arXiv preprint arXiv:1711.09784* (2017).
[78] Hehn, T. M., Kooij, J. F., and Hamprecht, F. A. "End-to-end learning of decision trees and forests". In: *International Journal of Computer Vision* 128.4 (2020), pp. 997–1011.
[79] Silva, A. et al. "Optimization Methods for Interpretable Differentiable Decision Trees Applied to Reinforcement Learning". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by S. Chiappa and R. Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 2020, pp. 1855–1865. URL:
[80] Tanno, R. et al. "Adaptive neural trees". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6166–6175.
[81] Wan, A. et al. "NBDT: Neural-backed decision trees". In: *arXiv preprint arXiv:2004.00221* (2020).
[82] Li, R.-H. and Belford, G. G. "Instability of decision tree classification algorithms". In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2002, pp. 570–575.
[83] Alvarez-Melis, D. and Jaakkola, T. S. "On the robustness of interpretability methods". In: *arXiv preprint arXiv:1806.08049* (2018).
[84] Yan, J. et al. "A Unified Framework for Decision Tree on Continuous Attributes". In: *IEEE Access* 7 (2019), pp. 11924–11933. DOI: 10.1109/ACCESS.2019.2892083.
[85] Li, J. et al. "Causal decision trees". In: *IEEE Transactions on Knowledge and Data Engineering* 29.2 (2016), pp. 257–271.
[86] Johansson, U. et al. "Interpretable regression trees using conformal prediction". In: *Expert systems with applications* 97 (2018), pp. 394–404.
[87] Carreira-Perpiñán, M. Á. and Hada, S. S. "Counterfactual Explanations for Oblique Decision Trees:Exact, Efficient Algorithms". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.8 (May 2021), pp. 6903–6911. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v35i8.16851.
[88] Goodman, B. and Flaxman, S. "European Union Regulations on Algorithmic Decision Making and a "Right to Explanation"". en. In: *AI Magazine* 38.3 (2017), pp. 50–57. ISSN: 0738-4602, 2371-9621. DOI: 10.1609/aimag.v38i3.2741.
[89] Confalonieri, R. et al. "A historical perspective of explainable artificial intelligence". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 11.1 (2021), e1391.
[90] Rudin, C. "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead". In: *Nature machine intelligence* 1.5 (2019), pp. 206–215.
[91] Stepin, I. et al. "A Survey of Contrastive and Counterfactual Explanation Generation Methods for Explainable Artificial Intelligence". In: *IEEE Access* 9 (2021), pp. 11974–12001. DOI: 10.1109/ACCESS.2021.3051315.
[92] Chou, Y.-L. et al. "Counterfactuals and causability in explainable artificial intelligence: Theory, algorithms, and applications". en. In: *Information Fusion* 81 (May 2022), pp. 59–83. ISSN: 15662535. DOI: 10.1016/j.inffus.2021.11.003.

[93] Doshi-Velez, F. and Kim, B. "Towards A Rigorous Science of Interpretable Machine Learning". en. In: arXiv:1702.08608 (Mar. 2017). URL:
[94] Hiabu, M., Meyer, J. T., and Wright, M. N. "Unifying local and global model explanations by functional decomposition of low dimensional structures". en. In: *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*. PMLR, Apr. 2023, pp. 7040–7060. URL:
[95] Aas, K., Jullum, M., and Løland, A. "Explaining individual predictions when features are dependent: More accurate approximations to Shapley values". en. In: arXiv:1903.10464 (Feb. 2020). arXiv:1903.10464 [cs, stat]. URL:
[96] Sotirou, T. et al. "Musiclime: Explainable multimodal music understanding". In: *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2025, pp. 1–5.
[97] Sundararajan, M., Dhamdhere, K., and Agarwal, A. "The Shapley Taylor Interaction Index". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by H. D. III and A. Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 9259–9268. URL:
[98] Datta, A., Sen, S., and Zick, Y. "Algorithmic Transparency via Quantitative Input Influence: Theory and Experiments with Learning Systems". en. In: *2016 IEEE Symposium on Security and Privacy (SP)*. San Jose, CA: IEEE, May 2016, pp. 598–617. ISBN: 978-1-5090-0824-7. DOI: 10.1109/SP.2016.42. URL:
[99] Menis Mastromichalakis, O. et al. "Semantic prototypes: Enhancing transparency without black boxes". In: *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 2024, pp. 1680–1688.
[100] Spanos, N. et al. "V-CECE: Visual Counterfactual Explanations via Conceptual Edits". In: *arXiv preprint arXiv:2509.16567* (2025).
[101] Filandrianos, G. et al. "Conceptual Edits as Counterfactual Explanations." In: *AAAI Spring Symposium: MAKE*. 2022.
[102] Mastromichalakis, O. M. et al. "Rule-based explanations of machine learning classifiers using knowledge graphs". In: *Proceedings of the AAAI Symposium Series*. Vol. 3. 1. 2024, pp. 193–202.
[103] Liartis, J. et al. "Semantic Queries Explaining Opaque Machine Learning Classifiers." In: *DAO-XAI*. 2021.
[104] Liartis, J. et al. "Searching for explanations of black-box classifiers in the space of semantic queries". In: *Semantic Web* 15.4 (2024), pp. 1085–1126.
[105] Lymperaiou, M. et al. "Halcece: A framework for explainable hallucination detection through conceptual counterfactuals in image captioning". In: *World Conference on Explainable Artificial Intelligence*. Springer. 2025, pp. 87–111.
[106] Ribeiro, M. T., Singh, S., and Guestrin, C. "Anchors: High-precision model-agnostic explanations". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.
[107] Guidotti, R. et al. "A survey of methods for explaining black box models". In: *ACM computing surveys (CSUR)* 51.5 (2018), pp. 1–42.
[108] Adebayo, J. et al. "Sanity Checks for Saliency Maps". In: arXiv:1810.03292 (Nov. 2020). arXiv:1810.03292 [cs]. DOI: 10.48550/arXiv.1810.03292. URL:
[109] Fong, R. and Vedaldi, A. "Interpretable Explanations of Black Boxes by Meaningful Perturbation". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. arXiv:1704.03296 [cs]. Oct. 2017, pp. 3449–3457. DOI: 10.1109/ICCV.2017.371. URL:
[110] Tsirtsis, S., De, A., and Gomez-Rodriguez, M. "Counterfactual Explanations in Sequential Decision Making Under Uncertainty". en. In: arXiv:2107.02776 (Oct. 2021). arXiv:2107.02776 [cs, stat]. URL:
[111] Lewis, D. K. *Counterfactuals*. Malden, Mass.: Blackwell, 1973.
[112] Miller, D. "Norm Theory: Comparing Reality to Its Alternatives". In: *Psychological review* (Jan. 1986). URL:
[113] Virgolin, M. and Fracaros, S. "On the robustness of sparse counterfactual explanations to adverse perturbations". In: *Artificial Intelligence* 316 (2023), p. 103840.
[114] Mastromichalakis, O. M., Liartis, J., and Stamou, G. "Beyond One-Size-Fits-All: How User Objectives Shape Counterfactual Explanations". In: (2025).
[115] Mastromichalakis, O. M., Liartis, J., and Stamou, G. "Beyond One-Size-Fits-All: Adapting Counterfactual Explanations to User Objectives". In: *arXiv preprint arXiv:2404.08721* (2024).

[116] Karimi, A.-H. et al. “A Survey of Algorithmic Recourse: Contrastive Explanations and Consequential Recommendations”. In: *ACM Comput. Surv.* 55.5 (Dec. 2022), 95:1–95:29. ISSN: 0360-0300. DOI: 10.1145/3527848.

[117] Pawelczyk, M., Broelemann, K., and Kasneci, G. “On Counterfactual Explanations under Predictive Multiplicity”. In: arXiv:2006.13132 (2020). DOI: 10.48550/arXiv.2006.13132. URL:

[118] Ley, D., Mishra, S., and Magazzeni, D. “GLOBE-CE: A Translation-Based Approach for Global Counterfactual Explanations”. In: arXiv:2305.17021 (Dec. 2023). arXiv:2305.17021 [cs]. DOI: 10.48550/arXiv.2305.17021. URL:

[119] Kavouras, L. et al. “GLANCE: Global Actions in a Nutshell for Counterfactual Explainability”. In: arXiv:2405.18921 (Oct. 2024). arXiv:2405.18921 [cs]. DOI: 10.48550/arXiv.2405.18921.

[120] Ribeiro, M. T., Singh, S., and Guestrin, C. ““Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: arXiv:1602.04938 (Aug. 2016). DOI: 10.48550/arXiv.1602.04938. URL:

[121] Brita, C. E., Linden, J. G. M. v. d., and Demirović, E. “Optimal Classification Trees for Continuous Feature Data Using Dynamic Programming with Branch-and-Bound”. In: arXiv:2501.07903 (Jan. 2025). arXiv:2501.07903 [cs]. DOI: 10.48550/arXiv.2501.07903. URL: