Optimality of the Greedy Local Explanations of EBaMl

A proof checked in Lean 4

Summary. EBaMl explains one prediction of a binary, multiclass or regression Explainable Boosting Machine (EBM) in two exact ways. The *minimum change required* is the fewest features to change so that the prediction flips. The *minimum sufficient reason* is the fewest features to fix so that no values of the other features can flip the prediction. Both queries come from Barceló, Monet, Pérez and Subercaseaux [1]. The code in `src/ebaml/local/` solves both with one sort. This paper proves that the sort is optimal. A regressor reduces to the binary case by a shift of the intercept, and the answer "inside an interval" has two margins. The counterfactual of two margins is the smaller of two exact answers. The sufficient reason of two margins is a covering problem. The code solves it with a MIP solver, a local search or a greedy, and the paper states what a result proves. A multiclass classifier has one margin for each rival class, so it uses the same results. The sufficient reason of several margins is NP-hard, because it contains Set Cover, and it has the same hardness of approximation. A Lean 4 file [2] checks every statement except two facts about running time and the statements about the solvers (Section 9, Section 10), and the appendix tells the reader how to repeat the check.

1 Claims and scope

The proof covers a binary EBM, a multiclass EBM and a regressor, all without interaction terms. The score of such a model is a sum of one term per feature, and this is the only fact that the proof needs.

Claim 1 (minimum change required). The function `_counterfactual` returns a set of features whose change flips the prediction. No set with fewer features has a change that flips the prediction. If the function returns `None`, no change of any set of features flips the prediction.

Claim 2 (minimum sufficient reason). The function `_sufficient_reason`, with the order of `fixing_order`, returns a set of features that is sufficient. No set with fewer features is sufficient. If the function returns `None`, no set of features is sufficient, and the set of all features is not sufficient either.

Claim 3 (regressor). For a regressor with a threshold τ, the margin of the decision is the margin of the binary model with the intercept $b - \tau$. Claims 1 and 2 therefore hold for the decisions "above τ" and "at most τ" with no new proof (Section 6, Lemmas 7 and 8). The code does this shift in `compute_local_facts`.

Claim 4 (inside an interval). When the predicted value is inside the interval, the decision has two margins. The fewest features that flip it are the fewest of the two answers for the margins, and `_smallest_counterfactual` returns this number (Theorem 6).

Claim 5 (solvers). For a decision with several margins, the code builds a covering problem, and a `CoveringSolver` of `covering_solvers.py` solves it: `MipCover`, `LocalSearchCover` or `GreedyCover`. Each set that a solver returns is sufficient, because the code checks it with the definition of Section 5. The status `minimality` says that the set is the smallest only if the MIP solver reports a proved optimum (`SOLVER_PROOF`), or if the size of the set equals a lower bound (`AT_LOWER_BOUND`) (Theorem 5, Corollary 3, Lemma 17). A proved optimum rests on the certificate of the MIP solver, which Lean does not check, and on floating-point tolerances (Section 9). The removal pass of the greedy returns a set from which no feature can be removed (Theorem 4).

Claim 6 (multiclass classifier). For a multiclass classifier, the predicted class w has one margin for each rival class j: the score of w minus the score of j, with the deviation $\sigma_w + \sigma_j$. The fewest features that change the predicted class are the fewest of the answers for the rivals, and `_smallest_counterfactual` returns this number (Theorem 7). A set keeps the class iff it is sufficient for the margin of every rival, so Claim 5 holds for the sufficient reason of a multiclass classifier (Lemma 11). The sum $\sigma_w + \sigma_j$ is an upper bound of the deviation of the margin for any correlation between the two scores (Lemma 12).

Claim 7 (hardness). The covering problem of several margins contains Set Cover. A set of features is sufficient for the problem of the reduction iff the matching sets cover every element (Theorem 8). A multiclass classifier with one rival class for each element has this problem as its sufficient reason (Theorem 9). The decision problem is NP-complete (Corollary 4), so an algorithm that finds a smallest sufficient set in polynomial time exists only if P = NP. The reduction keeps the optimum and the approximation ratio (Theorem 10). So the hardness of approximation of Set Cover [3] holds for the sufficient reason (Corollary 5). The claim covers the gain table of the sufficient reason. It does not cover the counterfactual of a chosen class (Section 10).

The proof does not cover the other parts of the code. Section 10 lists them. The counterfactual and the sufficient reason move different scores by z deviations. Section 10 states the uncertainty model of each claim.

2 Model

This section fixes the model and the glossary. The model follows the code line by line. The glossary gives one term to each concept, and the text uses this term everywhere.

Glossary.

Term	Meaning
feature	An element i of the finite set I. The set I has n elements.
bin	A value class of a feature. Feature i has the set of bins B_i. A categorical value is one bin.
score	The mean score $s_i(v)$ of bin v of feature i. It is a real number.
deviation	The standard deviation $\sigma_i(v) \geq 0$ of the score of bin v.
plausible bin	A bin with enough training support. The set $P_i \subseteq B_i$ holds them.
instance	The input to explain. It has bin $c_i \in P_i$ for each feature i.
margin	The distance of the prediction from the decision boundary, seen from the predicted class.
change	A set S of features and a plausible bin w_i for each feature in S.
flip	A change *flips* the prediction when its robust margin is below zero.
fixed	A feature is *fixed* when its bin is known to be c_i.
sufficient	A set F of fixed features is *sufficient* when its worst-case margin is above zero.
drop, gain	The numbers d_i and h_i of Section 2. The code sorts them.
threshold	The score τ at which a margin changes sign. It is 0 for a classifier.
decision	The claim that the explanations keep. A decision holds while every one of its margins is positive.

The code reads a model with one term per feature, and it rejects interaction terms (`read_features`). The model is therefore the data below. A bin set B_i may be any set. Only P_i must be finite, and it is, because the code stores one flag per bin.

Notation 1. (data). The finite set I of features. For each feature i: the set B_i, the score $s_i : B_i \to \mathbb{R}$, the deviation $\sigma_i : B_i \to \mathbb{R}$ with $\sigma_i \geq 0$, the finite set $P_i \subseteq B_i$ of plausible bins, and the bin $c_i \in P_i$ of the instance. The intercept $b \in \mathbb{R}$. The number $z \geq 0$ of deviations. The code takes z from `confidence_level` in $[0, 1)$ with the inverse normal distribution, so $z \geq 0$.

The code marks c_i as plausible for each feature (in `read_features`). Therefore P_i is not empty, and a minimum over P_i exists.

Notation 2. (sign and margin). The logit is $L = b + \sum_{i \in I} s_i(c_i)$. The sign is $\varepsilon = 1$ if $L > 0$, and $\varepsilon = -1$ otherwise. The signed score is $t_i(v) = \varepsilon \cdot s_i(v)$. The margin is

$$M = \varepsilon b + \sum_{i \in I} t_i(c_i) = \varepsilon L.$$

The sign is -1 for $L = 0$. This is the same as in the code (`sign = 1 if logit > 0 else -1`).

Notation 3. (pessimistic and guaranteed score). For $v \in P_i$ let $p_i(v) = t_i(v) + z\sigma_i(v)$ and $g_i(v) = t_i(v) - z\sigma_i(v)$. The score $p_i(v)$ is the highest signed score of bin v within z deviations. It is the hardest case for a flip. The score $g_i(v)$ is the lowest signed score within z deviations. It is the hardest case for the prediction to stay.

The code gives the value $+\infty$ to each bin outside P_i (in `robust_scores`). A minimum over all bins is then a minimum over P_i. The paper restricts the minimum to P_i, and so it needs no extended reals.

Notation 4. (change and robust margin). A *change* is a set $S \subseteq I$ and a bin $w_i \in P_i$ for each $i \in S$. Its robust margin is

$$R(S, w) = \varepsilon b + \sum_{i \notin S} t_i(c_i) + \sum_{i \in S} p_i(w_i).$$

The change *flips* the prediction iff $R(S, w) < 0$.

A feature outside S keeps its score without deviation. The code does this (in `_counterfactual`), and the module docstring says so. Section 10 compares this uncertainty model with the model of the sufficient reason.

Notation 5. (drop). $d_i = t_i(c_i) - \min_{v \in P_i} p_i(v)$. Let $\beta_i \in P_i$ be a bin where p_i reaches this minimum.

Notation 6. (worst-case margin and gain). For a set $F \subseteq I$ of fixed features,

$$W(F) = \varepsilon b + \sum_{i \in F} g_i(c_i) + \sum_{i \notin F} \min_{v \in P_i} g_i(v).$$

The set F is *sufficient* iff $W(F) > 0$. The gain of feature i is $h_i = g_i(c_i) - \min_{v \in P_i} g_i(v)$. Then $h_i \geq 0$, because $c_i \in P_i$.

Notation 7. (greedy rule). Let $e : \{0, ..., n-1\} \to I$ be a bijection such that the values $d_{e(0)}, ..., d_{e(n-1)}$ do not increase (for the gains, the values $h_{e(j)}$). Ties may have any order. Let Π_k be the sum of the first k values. The code returns the first k with $\Pi_k > M$ for the drops (the search starts at $k = 1$), and the first k with $W(\varnothing) + \Pi_k > 0$ for the gains (the search starts at $k = 0$). If no $k \leq n$ has this property, the code returns `None`.

Correspondence of symbols to the code. The code is in `src/ebaml/local/facts.py`, `fact_features.py`, `counterfactuals.py`, `sufficient_reasons.py` and `completions.py`. The table names the function that computes each symbol.

Symbol	Computed by	Function
z	`statistics.NormalDist().inv_cdf((1 + confidence_level) / 2)`	`compute_local_facts`
b	`intercept = float(np.ravel(ebm.intercept_)[0])`	`compute_local_facts`
L	`logit = intercept + sum(f.current ...)`	`compute_local_facts`
ε	`sign = 1 if logit > 0 else -1`	`compute_local_facts`
$t_i(v)$	`scores = sign * feature.scores`	`robust_scores`
P_i, $c_i \in P_i$	`_plausible_bins`, then `plausible[bin_index] = True`	`_plausible_bins, read_features`
p_i, g_i	`scores + stds` and `scores - stds`, with `np.inf` outside P_i	`robust_scores`

M	`margin = sign * intercept + sum(sign * f.current ... )`	`_counterfactual`
$\min p_i, \beta_i$	`best = int(np.argmin( pessimistic))`	`_counterfactual`
d_i	`sign * feature.current - float(pessimistic[best])`	`_counterfactual`
order e for d	`sorted(drops, ..., reverse=True)`	`_counterfactual`
first k with $\Pi_k > M$	`np.flatnonzero(margin - np.cumsum(...) < 0), flips[0] + 1`	`_counterfactual`
h_i	`guaranteed[feature.bin_ index] - guaranteed.min()`	`fixing_order`
order e for h	`sorted(gains, ..., reverse=True)`	`fixing_order`
$W(\varnothing)$	`worst = sign * intercept + sum(... .min() ...)`	`_sufficient_reason`
$W(\varnothing) + \Pi_k$	`margins = worst + np. concatenate(([0.0], np. cumsum(...)))`	`_sufficient_reason`
first k with $W(\varnothing) + \Pi_k > 0$	`np.flatnonzero(margins > 0), enough[0]`	`_sufficient_reason`

The Lean file `proofs/EBaMlProofs/LocalExplanations/Model.lean` states these notations as the structure `EBaMlProofs.LocalExplanations.Model` and the definitions `logit`, `eps`, `t`, `margin`, `p`, `g`, `pmin`, `drop`, `gain`, `beta`, `robustMargin`, `Admissible`, `CanFlip`, `worst` and `Sufficient`.

3 The abstract result

This section knows nothing about EBMs. Let I be a finite set with n elements, and let $d : I \to \mathbb{R}$. For a finite set S, write $d(S) = \sum_{i \in S} d(i)$.

Definition 1. (top set). A set $T \subseteq I$ is a *top set* of d iff $d(j) \leq d(i)$ for each $i \in T$ and each $j \notin T$.

Lean: `EBaMlProofs.LocalExplanations.Greedy.IsTopSet`, in `proofs/EBaMlProofs/LocalExplanations/Greedy.lean`.

Lemma 1. (top-set bound). *If T is a top set of d and $|S| = |T|$, then $d(S) \leq d(T)$.*

Lean: `EBaMlProofs.LocalExplanations.Greedy.topSet_sum_le`, in `proofs/EBaMlProofs/LocalExplanations/Greedy.lean`.

Warning. The condition $|S| = |T|$ is necessary. Take $d = (5, -1)$, $T = I$ and $S = \{1\}$. Then T is a top set, but $d(S) = 5 > 4 = d(T)$.

ASSUME: T is a top set of d, and $|S| = |T|$. **PROVE:** $d(S) \leq d(T)$.
⟨1⟩1 $d(S) - d(T) = d(S \setminus T) - d(T \setminus S)$, because both sets contain the part $S \cap T$.
⟨1⟩2 $|S \setminus T| = |T \setminus S|$, because $|S| = |T|$ and both differences remove $|S \cap T|$.
⟨1⟩3 **Case** $S \setminus T = \varnothing$.
  ⟨2⟩1 $T \setminus S = \varnothing$ by ⟨1⟩2.
  ⟨2⟩2 **Q.E.D.** Both differences are empty sums, so $d(S) - d(T) = 0$ by ⟨1⟩1.
⟨1⟩4 **Case** $S \setminus T \neq \varnothing$. Choose $i_0 \in S \setminus T$ with $d(i_0)$ largest in $S \setminus T$.
  ⟨2⟩1 $d(i) \leq d(i_0)$ for each $i \in S \setminus T$, by the choice of i_0.
  ⟨2⟩2 $d(i_0) \leq d(j)$ for each $j \in T \setminus S$, because $j \in T$, $i_0 \notin T$, and Definition 1.

⟨2⟩3 $d(S \setminus T) \le |S \setminus T| \cdot d(i_0) = |T \setminus S| \cdot d(i_0) \le d(T \setminus S)$, by ⟨2⟩1, ⟨1⟩2 and ⟨2⟩2.
⟨2⟩4 **Q.E.D.** by ⟨1⟩1 and ⟨2⟩3.
⟨1⟩5 **Q.E.D.** by ⟨1⟩3 and ⟨1⟩4.

The proof compares each value of $S \setminus T$ with one common bound $d(i_0)$. This replaces the pairing of the two sets by a bijection. It needs no choice of a bijection and gives the same bound.

Lemma 2. (prefixes are top sets). *Let $e : \{0, \dots, n-1\} \to I$ be a bijection such that $d \circ e$ does not increase. Let $T_k = e(\{0, \dots, k-1\})$ and $\Pi_k = \sum_{j<k} d(e(j))$. For $k \le n$, the set T_k is a top set of d, $|T_k| = k$, and $d(T_k) = \Pi_k$.*

Lean: `EBaMlProofs.LocalExplanations.Greedy.isTopSet_prefixSet`, `EBaMlProofs.LocalExplanations.Greedy.card_prefixSet`, `EBaMlProofs.LocalExplanations.Greedy.sum_prefixSet`, in `proofs/EBaMlProofs/LocalExplanations/Greedy.lean`.

ASSUME: e is a bijection, $d \circ e$ does not increase, and $k \le n$. **PROVE:** the three claims.
⟨1⟩1 $|T_k| = k$, because e is injective.
⟨1⟩2 $d(T_k) = \Pi_k$, because e maps the positions $0, \dots, k-1$ one-to-one onto T_k.
⟨1⟩3 T_k is a top set.
⟨2⟩1 **Let** $i \in T_k$ and $j \notin T_k$. **Let** $a = e^{-1}(i)$ and $b = e^{-1}(j)$.
⟨2⟩2 $a < k \le b$, by the definition of T_k.
⟨2⟩3 $d(j) = d(e(b)) \le d(e(a)) = d(i)$, because $d \circ e$ does not increase and $a < b$.
⟨2⟩4 **Q.E.D.** by ⟨2⟩3.
⟨1⟩4 **Q.E.D.** by ⟨1⟩1, ⟨1⟩2 and ⟨1⟩3.

Theorem 1. (greedy threshold). *Let $\tau \in \mathbb{R}$, and let e be as in Lemma 2. **Assume** $k \le n$, $\Pi_k > \tau$, and $\Pi_j \le \tau$ for each $j < k$. Then (a) $d(T_k) > \tau$, and (b) each set S with $d(S) > \tau$ has $|S| \ge k$.*

Lean: `EBaMlProofs.LocalExplanations.Greedy.threshold_minimal`, in `proofs/EBaMlProofs/LocalExplanations/Greedy.lean`.

ASSUME: $k \le n$, $\Pi_k > \tau$, and $\Pi_j \le \tau$ for each $j < k$. **PROVE:** (a) and (b).
⟨1⟩1 (a) holds, because $d(T_k) = \Pi_k > \tau$ by Lemma 2.
⟨1⟩2 (b) holds. **Assume** $|S| < k$. **Prove** $d(S) \le \tau$.
⟨2⟩1 $|S| \le n$, so $T_{|S|}$ exists, and $|T_{|S|}| = |S|$ by Lemma 2.
⟨2⟩2 $d(S) \le d\big(T_{|S|}\big) = \Pi_{|S|}$ by Lemma 1, Lemma 2 and ⟨2⟩1.
⟨2⟩3 $\Pi_{|S|} \le \tau$ by the assumption, because $|S| < k$.
⟨2⟩4 **Q.E.D.** by ⟨2⟩2 and ⟨2⟩3.
⟨1⟩3 **Q.E.D.** by ⟨1⟩1 and ⟨1⟩2.

Corollary 1. (no answer). *If $\Pi_k \le \tau$ for each $k \le n$, then $d(S) \le \tau$ for each set S.*

Lean: `EBaMlProofs.LocalExplanations.Greedy.no_answer`, in `proofs/EBaMlProofs/LocalExplanations/Greedy.lean`.

Let S be a set. Then $|S| \le n$, and $d(S) \le d\big(T_{|S|}\big) = \Pi_{|S|} \le \tau$ by Lemma 1, Lemma 2 and the assumption.

Corollary 2. (chosen values are positive). *Under the assumptions of Theorem 1, $d(e(j)) > 0$ for each $j < k$.*

Lean: `EBaMlProofs.LocalExplanations.Greedy.pos_of_threshold`, in `proofs/EBaMlProofs/LocalExplanations/Greedy.lean`.

ASSUME: the assumptions of Theorem 1, and $j < k$ (so $k \ge 1$). **PROVE:** $d(e(j)) > 0$.
⟨1⟩1 $d(e(k-1)) = \Pi_k - \Pi_{k-1} > 0$, because $\Pi_{k-1} \le \tau < \Pi_k$.
⟨1⟩2 $d(e(j)) \ge d(e(k-1))$, because $j \le k-1$ and $d \circ e$ does not increase.
⟨1⟩3 **Q.E.D.** by ⟨1⟩1 and ⟨1⟩2.

4 Application to EBMs

This section applies Theorem 1 to the model of Section 2. The sum of all scores is additive over the features, and this is the reason why the abstract result applies.

Lemma 3. (the margin is not negative). $M \geq 0$. *Therefore no change with $S = \varnothing$ flips the prediction.*

Lean: `EBaMlProofs.LocalExplanations.Model.margin_nonneg`, in `proofs/EBaMlProofs/LocalExplanations/Model.lean`.

ASSUME: the model of Section 2. **PROVE:** $M \geq 0$.
⟨1⟩1 **Case** $L > 0$. Then $\varepsilon = 1$ and $M = L > 0$.
⟨1⟩2 **Case** $L \leq 0$. Then $\varepsilon = -1$ and $M = -L \geq 0$.
⟨1⟩3 **Q.E.D.** by the two cases, which cover all values of L.
The change $(\varnothing, w)$ has $R(\varnothing, w) = M \geq 0$. The code searches from $k = 1$, so it loses nothing.

Lemma 4. (best change of a set). *For each change (S, w): $R(S, w) = M - \sum_{i \in S}(t_i(c_i) - p_i(w_i)) \geq M - d(S)$, with equality when $w = \beta$. Therefore a set S has a change that flips the prediction iff $d(S) > M$.*

Lean: `EBaMlProofs.LocalExplanations.Model.robustMargin_eq`, `EBaMlProofs.LocalExplanations.Model.robustMargin_ge`, `EBaMlProofs.LocalExplanations.Model.robustMargin_beta`, `EBaMlProofs.LocalExplanations.Model.canFlip_iff`, in `proofs/EBaMlProofs/LocalExplanations/Model.lean`.

ASSUME: a change (S, w). **PROVE:** the equation, the bound, the equality and the equivalence.
⟨1⟩1 The equation holds. The sum in $R(S, w)$ differs from the sum in M only in the terms with $i \in S$. There $p_i(w_i)$ replaces $t_i(c_i)$.
⟨1⟩2 The bound holds. For $i \in S$ the bin w_i is in P_i, so $p_i(w_i) \geq \min_{v \in P_i} p_i(v)$. Hence $t_i(c_i) - p_i(w_i) \leq d_i$. Sum over S and use ⟨1⟩1.
⟨1⟩3 The equality holds for $w = \beta$, because $p_i(\beta_i) = \min_{v \in P_i} p_i(v)$, so each term equals d_i.
⟨1⟩4 **Case** a change (S, w) flips the prediction. Then $0 > R(S, w) \geq M - d(S)$ by ⟨1⟩2, so $d(S) > M$.
⟨1⟩5 **Case** $d(S) > M$. Then $R(S, \beta) = M - d(S) < 0$ by ⟨1⟩3, and β is a change because $\beta_i \in P_i$.
⟨1⟩6 **Q.E.D.** by ⟨1⟩4 and ⟨1⟩5.

Lemma 5. (fixing adds the gain). $W(F) = W(\varnothing) + \sum_{i \in F} h_i$. *Therefore F is sufficient iff $\sum_{i \in F} h_i > -W(\varnothing)$.*

Lean: `EBaMlProofs.LocalExplanations.Model.worst_eq`, `EBaMlProofs.LocalExplanations.Model.sufficient_iff`, in `proofs/EBaMlProofs/LocalExplanations/Model.lean`.

ASSUME: a set F. **PROVE:** the equation and the equivalence.
⟨1⟩1 For each feature i the term of $W(F)$ is $\min_{v \in P_i} g_i(v)$ if $i \notin F$, and $g_i(c_i) = \min_{v \in P_i} g_i(v) + h_i$ if $i \in F$.
⟨1⟩2 The equation holds. Sum ⟨1⟩1 over all features. The sum of the minima is the part of $W(\varnothing)$ without εb, and the extra terms are $\sum_{i \in F} h_i$.
⟨1⟩3 **Q.E.D.** The equivalence follows from the equation and the definition $W(F) > 0$.

Theorem 2. (minimum change). *Let e sort the drops d_i in descending order, and let k be the first position with $\Pi_k > M$ (so $k \leq n$, $\Pi_k > M$, and $\Pi_j \leq M$ for $1 \leq j < k$). Then (a) the change of the first k features to the bins β_i flips the prediction, (b) no change of fewer than k features flips the prediction, (c) each of these k features moves to a bin $\beta_i \neq c_i$, and (d) if $\Pi_k \leq M$ for each $k \leq n$, no change of any set of features flips the prediction.*

Lean: `EBaMlProofs.LocalExplanations.Model.min_change (a, b)`, `EBaMlProofs.LocalExplanations.Model.chosen_bin_differs (c)`, `EBaMlProofs.LocalExplanations.Model.min_change_none (d)`, in `proofs/EBaMlProofs/LocalExplanations/Model.lean`.

ASSUME: the model, an ordering e of the drops, and the conditions of the statement. **PROVE:** (a)–(d).
⟨1⟩1 $\Pi_0 = 0 \leq M$ by Lemma 3, so $\Pi_j \leq M$ for each $j < k$ including $j = 0$.
⟨1⟩2 Theorem 1 with d the drops and $\tau = M$ applies, by ⟨1⟩1.
⟨1⟩3 (a) holds. Theorem 1(a) gives $d(T_k) > M$, and Lemma 4 gives $R(T_k, \beta) = M - d(T_k) < 0$.
⟨1⟩4 (b) holds. A change of a set S that flips the prediction gives $d(S) > M$ by Lemma 4. Theorem 1(b) gives $|S| \geq k$.
⟨1⟩5 (c) holds. Let $j < k$ and $i = e(j)$.

⟨2⟩1 $d_i > 0$ by Corollary 2.
⟨2⟩2 **Assume** $\beta_i = c_i$. Then $\min_{v \in P_i} p_i(v) = p_i(c_i) = t_i(c_i) + z\sigma_i(c_i) \geq t_i(c_i)$, because $z \geq 0$ and $\sigma_i \geq 0$. Hence $d_i \leq 0$. This contradicts ⟨2⟩1.
⟨2⟩3 **Q.E.D.** by ⟨2⟩2.
⟨1⟩6 (d) holds. By Corollary 1, $d(S) \leq M$ for each set S. Lemma 4 then shows that no change of S flips the prediction.
⟨1⟩7 **Q.E.D.** by ⟨1⟩3 to ⟨1⟩6.

Theorem 3. (minimum sufficient reason). *Let e sort the gains h_i in descending order, and let k be the first position, counted from 0, with $W(\varnothing) + \Pi_k > 0$. Then (a) the first k features are sufficient, (b) no set of fewer than k features is sufficient, and (c) if $W(\varnothing) + \Pi_k \leq 0$ for each $k \leq n$, no set of features is sufficient. This includes the set of all features.*

Lean: `EBaMlProofs.LocalExplanations.Model.min_sufficient_reason (a, b)`, `EBaMlProofs.LocalExplanations.Model.min_sufficient_reason_none (c)`, in `proofs/EBaMlProofs/LocalExplanations/Model.lean`.

ASSUME: the model, an ordering e of the gains, and the conditions of the statement. **PROVE:** (a)–(c).
⟨1⟩1 Theorem 1 with d the gains and $\tau = -W(\varnothing)$ applies. The condition $W(\varnothing) + \Pi_k > 0$ means $\Pi_k > \tau$, and $W(\varnothing) + \Pi_j \leq 0$ for $j < k$ means $\Pi_j \leq \tau$. For $k = 0$ the second condition is empty.
⟨1⟩2 (a) holds. Theorem 1(a) gives $h(T_k) > -W(\varnothing)$, so T_k is sufficient by Lemma 5.
⟨1⟩3 (b) holds. A sufficient set F has $h(F) > -W(\varnothing)$ by Lemma 5, so $|F| \geq k$ by Theorem 1(b).
⟨1⟩4 (c) holds. By Corollary 1, $h(F) \leq -W(\varnothing)$ for each set F, so no set is sufficient by Lemma 5.
⟨1⟩5 **Q.E.D.** by ⟨1⟩2 to ⟨1⟩4.

The code returns the first k in each case, and `None` in the case of Theorem 2(d) and Theorem 3(c). Claims 1 and 2 of Section 1 follow from Theorems 2 and 3, provided that the code computes the quantities of Section 2. The table in Section 2 shows where.

5 Decisions with several margins

A decision such as "inside an interval" holds while two margins are positive at the same time. This section knows nothing about EBMs. A *problem* has a finite set I of features, a finite index set K of margins, a number W_k for each margin (the worst case when no feature is fixed), and a gain $h_{k,i} \geq 0$ for each margin k and feature i. A set $F \subseteq I$ is *sufficient* iff $W_k + \sum_{i \in F} h_{k,i} > 0$ for each $k \in K$. With one margin this is the definition of Section 2.

Lemma 6. (monotone). *If $F \subseteq G$ and F is sufficient, then G is sufficient.*

Lean: `EBaMlProofs.LocalExplanations.Covering.Problem.sufficient_mono`, in `proofs/EBaMlProofs/LocalExplanations/Covering.lean`.

Let $k \in K$. The gains are not negative, so $\sum_{i \in F} h_{k,i} \leq \sum_{i \in G} h_{k,i}$. Hence $W_k + \sum_{i \in G} h_{k,i} \geq W_k + \sum_{i \in F} h_{k,i} > 0$.

The greedy (`greedy_cover` in `covering_greedy.py`) has two passes. The first pass adds the feature with the largest gain of coverage (Section 9) until the set is sufficient. The second pass, *prune*, visits the added features from the last to the first. It removes a feature when the set stays sufficient without it. The proof allows any order in both passes. Let $\mathrm{prune}(l, C)$ be the result for a list l of features and a set C at the start of the second pass.

Theorem 4. (the removal pass). *Let C be sufficient, and let l hold every feature of C. Then $\mathrm{prune}(l, C)$ is a sufficient subset of C, and no feature can be removed from it: for each $f \in \mathrm{prune}(l, C)$, the set $\mathrm{prune}(l, C) \setminus \{f\}$ is not sufficient.*

Lean: `EBaMlProofs.LocalExplanations.Covering.Problem.prune_minimal`, `EBaMlProofs.LocalExplanations.Covering.Problem.prune_invariant`, in `proofs/EBaMlProofs/LocalExplanations/Covering.lean`.

ASSUME: a sufficient set C, and a list l that holds every feature of C. **PROVE:** the three claims. **Invariant** for a list l and a set C: C is sufficient, and each $f \in C$ is in l or is needed (that is, $C \setminus \{f\}$ is not sufficient). We show by induction on l that the invariant gives the three claims.

⟨1⟩1 **Case** l is empty. Then $\text{prune}(l, C) = C$, and each $f \in C$ is needed, because it is not in l.

⟨1⟩2 **Case** $l = f_0 :: l'$. Let C' be $C \setminus \{f_0\}$ if this set is sufficient, and C otherwise. The pass continues with C' and l'.

⟨2⟩1 C' is sufficient, and $C' \subseteq C$, by the choice of C'.

⟨2⟩2 Each $f \in C'$ is in l' or is needed in C'.

⟨3⟩1 **Case** f is in l'. Done.

⟨3⟩2 **Case** $f = f_0$. Then $f_0 \in C'$, so the first choice was not taken, $C' = C$, and $C \setminus \{f_0\}$ is not sufficient. Hence f_0 is needed in C'.

⟨3⟩3 **Case** f is needed in C. Then $C' \setminus \{f\} \subseteq C \setminus \{f\}$. If $C' \setminus \{f\}$ were sufficient, then $C \setminus \{f\}$ would be sufficient by Lemma 6. So f is needed in C'.

⟨3⟩4 **Q.E.D.** The three cases cover each $f \in C'$, because $f \in C$ and the invariant holds for C.

⟨2⟩3 The invariant holds for C' and l' by ⟨2⟩1 and ⟨2⟩2. The induction hypothesis gives the three claims for the result, and the result is a subset of $C' \subseteq C$.

⟨1⟩3 **Q.E.D.** The assumption of the theorem is the invariant for C, because each $f \in C$ is in l.

A MIP solver that ends with the status `OPTIMAL` claims two facts: its set F is sufficient, and no set with fewer features is sufficient. The next theorem is the form of this claim that the code uses. A search by size, as the brute-force oracle of the tests does, visits at most 2^n sets. The code does not search by size.

Theorem 5. (the exact search). *Let F be sufficient, and let no set with fewer than $|F|$ features be sufficient. Then each sufficient set has at least $|F|$ features. A search by size visits at most 2^n sets for n features.*

Lean: `EBaMlProofs.LocalExplanations.Covering.Problem.exact_minimum`, `EBaMlProofs.LocalExplanations.Covering.Problem.card_search_space`, in `proofs/EBaMlProofs/LocalExplanations/Covering.lean`.

A sufficient set G with $|G| < |F|$ would contradict the assumption. The sets that a search by size visits are subsets of I, and there are 2^n of them.

Corollary 3. (a set of at most one feature). *Let R be sufficient with $|R| \leq 1$. If R is empty, or the empty set is not sufficient, then each sufficient set has at least $|R|$ features.*

Lean: `EBaMlProofs.LocalExplanations.Covering.Problem.minimum_of_card_le_one`, in `proofs/EBaMlProofs/LocalExplanations/Covering.lean`.

If R is empty, the claim is $0 \leq |G|$. Otherwise the empty set is not sufficient, so each sufficient set G is not empty, and $|G| \geq 1 \geq |R|$.

Each solver of the code checks the empty set first. When the empty set is not sufficient, the lower bound of the solver is at least 1: the code takes `max(1, ...)` of the LP value in `_lower_bound`, and it uses 1 when the solver has no backend for the LP. So a result with one feature reaches the bound, and its status `minimality` is `AT_LOWER_BOUND`. Corollary 3 proves this case. For a larger result, the status rests on the solver or on the bound of Lemma 17. An example shows that the two passes of the greedy alone do not give a smallest set. Take three margins with $W = \left(-\frac{3}{2}, -\frac{5}{2}, -\frac{7}{2}\right)$ and four features with the gains $h_1 = (2, 2, 0)$, $h_2 = (2, 0, 2)$, $h_3 = (1, 2, 3)$ and $h_4 = (0, 3, 3)$. The first pass adds the features $3, 1, 2$, and the set is sufficient. No feature can be removed, but $\{2, 4\}$ is also sufficient and has fewer features. The test problem `GREEDY_IS_NOT_OPTIMAL` is this example.

6 Regressors and the answer inside an interval

A regressor predicts the value $L = b + \sum_i s_i(c_i)$. A decision is a claim about L. The code gives it as a list of *margins* (`Margin` in `decisions.py`). Each margin has a sign $\sigma \in \{1, -1\}$ and a threshold τ, and its value is $\sigma(L - \tau)$. A binary classifier is the case with one margin and $\tau = 0$. The code (`Threshold.side` and `Interval.side`) chooses the margins as follows.

Claim that holds	Margins (σ, τ)
above t	$(1, t)$
at most t	$(-1, t)$
above l and at most u (inside)	$(1, l)$ and $(-1, u)$

The next lemmas apply to the margin with the data (σ, τ) when σ is the sign that the claim has. Let m_τ be the model of Section 2 with the intercept $b - \tau$ and the same scores, deviations, plausible bins and instance bins. The code does this shift: it passes `intercept - margin.threshold` to the algorithms (`Bound` in `compute_local_facts`).

Lemma 7. (the shift). *(a) The logit of m_τ is $L - \tau$. (b) The sign of m_τ is 1 if $\tau < L$, and -1 otherwise. (c) The margin of m_τ is $\varepsilon(L - \tau)$ with this sign. So a prediction above τ has the margin $L - \tau$ of the claim "above τ", and any other prediction has the margin $\tau - L$ of the claim "at most τ".*

Lean: `EBaMlProofs.LocalExplanations.Model.shift_logit`, `EBaMlProofs.LocalExplanations.Model.shift_eps`, `EBaMlProofs.LocalExplanations.Model.shift_margin_eq`, in `proofs/EBaMlProofs/LocalExplanations/Regressor.lean`.

(a) $(b - \tau) + \sum_i s_i(c_i) = L - \tau$. (b) The sign is 1 iff the logit $L - \tau$ is above 0, which holds iff $\tau < L$. (c) The margin is the sign times the logit (Notation of Section 2), and the logit is $L - \tau$.

The shifted model m_τ is a model of Section 2. Theorems 2 and 3 therefore hold for it, and they are the statements about the regressor decision "above τ" or "at most τ" (Claim 3). The function `_counterfactual` and the function `_sufficient_reason` run on m_τ.

Lemma 8. (a flip crosses the threshold). *Let $z = 0$, and let x be the predicted value after a change (S, w). The robust margin $R(S, w)$ of m_τ is negative iff $x < \tau$ when $\tau < L$, and iff $x > \tau$ when $L \le \tau$.*

Lean: `EBaMlProofs.LocalExplanations.Model.shift_robustMargin_neg_iff`, in `proofs/EBaMlProofs/LocalExplanations/Regressor.lean`.

With $z = 0$ the pessimistic score is the signed score, so $R(S, w) = \varepsilon(x - \tau)$ for the sign ε of m_τ. For $\varepsilon = 1$ this is negative iff $x < \tau$. For $\varepsilon = -1$ it is negative iff $x > \tau$.

A change that the code finds therefore moves the value to the other side of τ. The converse fails only at the point $x = \tau$ for a prediction above τ. Section 10 discusses this tie.

When the predicted value is inside the interval $(l, u]$, the two shifted models have the signs 1 and -1. A change flips the decision "inside" iff it makes one of the two robust margins negative.

Theorem 6. (inside an interval). *Let $l < L \le u$. Let k_1 and k_2 be the positions of Theorem 2 for the models m_l and m_u. Then (a) the signs of m_l and m_u are 1 and -1, (b) a set S flips the decision iff it flips the margin of m_l or the margin of m_u, (c) the smallest set that flips the decision has $\min(k_1, k_2)$ features, and (d) if neither model has a position, no set flips the decision.*

Lean: `EBaMlProofs.LocalExplanations.Model.inside_eps (a)`, `EBaMlProofs.LocalExplanations.Model.canFlipInside_iff (b)`, `EBaMlProofs.LocalExplanations.Model.min_change_inside (c)`, `EBaMlProofs.LocalExplanations.Model.min_change_inside_none (d)`, in `proofs/EBaMlProofs/LocalExplanations/Regressor.lean`.

ASSUME: $l < L \le u$ and the positions k_1 and k_2 of Theorem 2. **PROVE:** (a)–(d).

⟨1⟩1 (a) holds by Lemma 7(b), because $l < L$ and $\neg(u < L)$.

⟨1⟩2 (b) holds. A change (S, w) flips the decision iff $R_l(S, w) < 0$ or $R_u(S, w) < 0$. The plausible bins and the set S are the same for both models. So the change flips one margin iff S can flip that margin.

⟨1⟩3 (c) holds.

⟨2⟩1 The first $\min(k_1, k_2)$ features of the model with the smaller position, moved to the bins β, flip the decision, by Theorem 2(a) and ⟨1⟩2.

⟨2⟩2 A set S that flips the decision has $|S| \geq k_1$ or $|S| \geq k_2$ by Theorem 2(b) and ⟨1⟩2. So $|S| \geq \min(k_1, k_2)$.
⟨2⟩3 **Q.E.D.** by ⟨2⟩1 and ⟨2⟩2.
⟨1⟩4 (d) holds by Theorem 2(d) for each model and ⟨1⟩2.
⟨1⟩5 **Q.E.D.** by ⟨1⟩1 to ⟨1⟩4.

The code takes the first of the two answers when both have the same size (`min` in `_smallest_counterfactual`). The size is the same, and the named features may differ.

The sufficient reason of a decision with the margins m_k is a covering problem.

Lemma 9. (several margins). *Let m_k be the shifted models of the margins of a decision. A set F is sufficient for the problem with $W_k = W_k(\varnothing)$ and the gains $h_{k,i}$ of m_k iff F is sufficient for each model m_k.*

Lean: `EBaMlProofs.LocalExplanations.problemOfModels_sufficient`, in `proofs/EBaMlProofs/LocalExplanations/Regressor.lean`.

By Lemma 5, F is sufficient for m_k iff $W_k(\varnothing) + \sum_{i \in F} h_{k,i} > 0$. This is the condition of the problem for the margin k. The gains of a model are not negative, so the data form a problem.

Lemma 9 puts the sufficient reason of "inside" under Lemma 6, Theorems 4 and 5 and Corollary 3. The function `gain_table` computes the numbers W_k and $h_{k,i}$, and `_several_margins_reason` gives them to the solver of `LocalOptions.search` as a problem with one alternative for each feature.

Correspondence of symbols to the code for this section.

Symbol	Computed by	Function
(σ, τ)	`Margin(sign, threshold, opposite)`	`Threshold.side`, `Interval.side`
intercept $b - \tau$	`Bound(intercept - margin. threshold, margin.sign)`	`compute_local_facts`
the answer for a margin	`_counterfactual(features, intercept - margin.threshold, margin.sign, ...)`	`_smallest_counterfactual`
$\min(k_1, k_2)$	`min(found, key=lambda c: len( c.changes), default=None)`	`_smallest_counterfactual`
$W_k(\varnothing)$	`bound.sign * bound.intercept + sum(scores.min() ...)`	`gain_table`
$h_{k,i}$	`scores[f.bin_index] - scores. min()`	`gain_table`
first pass of `Greedy`	`_next_pick` until `problem. sufficient(picked)`	`greedy_cover`
prune	`for pick in reversed(list( picked))`	`greedy_cover`
F is sufficient	`all(margin > 0 for margin in self.margins(choices))`	`CoveringProblem.sufficient`
the bound 1 of Corollary 3	`max(1, math.ceil(...))`, or `1` with no backend	`_lower_bound`, `_open`
the status `minimality`	`len(ordered) <= lower_bound`, then `proved`	`make_cover`

7 Multiclass classifiers

A multiclass EBM has one score $s_{k,i}(v)$ for each class k, each feature i and each bin v, and one intercept b_k for each class. The score of the class k at the instance is $S_k = b_k + \sum_i s_{k,i}(c_i)$. The predicted class w

has the highest score. `MulticlassLocal.frame` gives it one margin for each rival $j \neq w$: the sign is 1, the threshold is 0, and `Margin.columns` names the pair (w, j). The code reads the scores, the intercepts and the deviations of the pair in `Feature.against`, `Margin.offset` and `Margin.value`.

Let m_j be the model of Section 2 with the intercept $b_w - b_j$, the scores $s_{w,i} - s_{j,i}$, the deviations $\sigma_{w,i} + \sigma_{j,i}$, and the same plausible bins and instance bins. The deviation of a difference of two scores has no covariance term, because an EBM stores no covariance between the scores of two classes.

Lemma 10. (the winner). *(a) The logit of m_j is $S_w - S_j$. (b) Every logit of m_j with $j \neq w$ is positive iff $S_j < S_w$ for every rival. (c) When the logit of m_j is positive, the sign of m_j is* 1.

Lean: `EBaMlProofs.LocalExplanations.Multiclass.logit_marginModel (a)`, `EBaMlProofs.LocalExplanations.Multiclass.winner_iff (b)`, `EBaMlProofs.LocalExplanations.Multiclass.marginModel_eps (c)`, in `proofs/EBaMlProofs/LocalExplanations/Multiclass.lean`.

(a) $\big(b_w - b_j\big) + \sum_i \big(s_{w,i}(c_i) - s_{j,i}(c_i)\big) = S_w - S_j$. (b) follows from (a). (c) The sign is 1 iff the logit is above 0.

Lemma 10 shows that the models m_j are the margins of the code, and it holds only when the predicted class wins every pair strictly. Section 10 discusses the tie.

Theorem 7. (the class changes). *Let $S_w > S_j$ for every rival j. Let a set S* change the class *if some admissible change of S makes the robust margin of some m_j negative. Then (a) S changes the class iff it can flip the margin of m_j for some rival j. Let the rival j_0 have the position k of Theorem 2, and let every other rival either have no set that flips its margin, or need at least k features. Then (b) the set of the first k features of j_0, moved to the bins β, changes the class, and no set of fewer than k features changes it. (c) If no set can flip the margin of any rival, no set changes the class.*

Lean: `EBaMlProofs.LocalExplanations.Multiclass.canFlipClass_iff (a)`, `EBaMlProofs.LocalExplanations.Multiclass.min_change_class (b)`, `EBaMlProofs.LocalExplanations.Multiclass.min_change_class_none (c)`, in `proofs/EBaMlProofs/LocalExplanations/Multiclass.lean`.

ASSUME: $S_w > S_j$ for every rival, and the positions of Theorem 2. **PROVE:** (a)–(c).

⟨1⟩1 (a) holds. A change (S, v) changes the class iff $R_j(S, v) < 0$ for some rival j. The plausible bins and the set S are the same for every rival, so the change flips the margin of m_j iff S can flip it.

⟨1⟩2 (b) holds.

⟨2⟩1 The first k features of j_0, moved to the bins β, flip the margin of m_{j_0} by Theorem 2(a). So they change the class by ⟨1⟩1.

⟨2⟩2 A set S that changes the class can flip the margin of some rival j by ⟨1⟩1. By the assumption on the rivals, either this is impossible, or $|S| \geq k$.

⟨2⟩3 **Q.E.D.** by ⟨2⟩1 and ⟨2⟩2.

⟨1⟩3 (c) holds by ⟨1⟩1.

⟨1⟩4 **Q.E.D.** by ⟨1⟩1 to ⟨1⟩3.

Each rival has its own position k_j from Theorem 2, or no position. The rival with the smallest position is j_0, and the other rivals have a larger position or none, so the assumption of Theorem 7 holds. The code takes the first rival when two rivals have the same position (`min` in `_smallest_counterfactual`). The size is the same, and the named features may differ.

The sufficient reason of the predicted class has one condition for each rival, so it is a covering problem.

Lemma 11. (several rivals). *A set F is sufficient for the covering problem of the rivals iff F is sufficient for each model m_j with $j \neq w$.*

Lean: `EBaMlProofs.LocalExplanations.Multiclass.sufficient_class_iff`, in `proofs/EBaMlProofs/LocalExplanations/Multiclass.lean`.

This is Lemma 9 for the family of the models m_j, indexed by the rivals.

Lemma 11 puts the sufficient reason of a multiclass classifier under Lemma 6, Theorems 4 and 5 and Corollary 3, as Lemma 9 does for the answer "inside". The function `gain_table` computes the numbers, and the search of `LocalOptions.search` solves the problem. The status `minimality` has the same meaning as in Claim 5.

Lemma 12. (the deviation of a margin). *Let $\sigma_x, \sigma_y \geq 0$ be the deviations of two scores, and let $\rho \in [-1, 1]$ be their correlation. The variance of their difference, $\sigma_x^2 + \sigma_y^2 - 2\rho\sigma_x\sigma_y$, is at most $\left(\sigma_x + \sigma_y\right)^2$.*

Lean: `EBaMlProofs.LocalExplanations.sd_sum_bound`, in `proofs/EBaMlProofs/LocalExplanations/Multiclass.lean`.

The difference of the two sides is $2(1 + \rho)\sigma_x\sigma_y \geq 0$.

Lemma 12 shows that the sum $\sigma_w + \sigma_j$ is at least the deviation of the margin for every correlation. The model of Section 2 takes the deviation of each margin as given, so the proof covers the code with this sum.

Correspondence of symbols to the code for this section.

Symbol	Computed by	Function
w	`int(np.argmax(scores))`	`MulticlassLocal.frame`
margin of m_j	`Margin(1, 0.0, self.classes[rival], (winner, rival))`	`MulticlassLocal.frame`
$b_w - b_j$	`intercepts[winner] - intercepts[rival]`	`Margin.offset`
$s_{w,i} - s_{j,i}$	`scores[:, winner] - scores[:, rival]`	`Feature.scores`
$\sigma_{w,i} + \sigma_{j,i}$	`stds[:, winner] + stds[:, rival]`	`Feature.stds`
the answer for a rival	`_counterfactual(... margin.offset(intercepts), margin.sign, ...)`	`_smallest_counterfactual`
the rival with the smallest position	`min(found, key=lambda c: len(c.changes), default=None)`	`_smallest_counterfactual`
$W_j(\varnothing)$, $h_{j,i}$	`robust_scores(f.against(bound.columns), ...)`	`gain_table`

8 The sufficient reason of several margins is NP-hard

Section 5 shows that sorting does not solve the covering problem. This section shows that no fast exact algorithm exists unless P = NP, which is why the code uses a MIP solver with a time limit. The covering problem contains Set Cover, so each exact algorithm for it is also an exact algorithm for Set Cover.

An instance of *Set Cover* has a finite set U of elements, a finite family of sets $S_i \subseteq U$ with an index $i \in I$, and a number K. The question is whether some $F \subseteq I$ with $|F| \leq K$ has $\bigcup_{i \in F} S_i = U$. Such an F is a *cover*. Karp [4] proves that this question is NP-complete.

The reduction makes one margin for each element and one feature for each set. For each $k \in U$ and $i \in I$, let $W_k = -\frac{1}{2}$, and let $h_{k,i} = 1$ if $k \in S_i$ and $h_{k,i} = 0$ otherwise. The gains are not negative, so the numbers form a problem in the sense of Section 5. The table has $|U|$ $|I|$ entries, so the reduction needs a time that is linear in the size of the instance.

Lemma 13. (counting the gains). *Let $k \in U$ and $F \subseteq I$. Then $\sum_{i \in F} h_{k,i} > \frac{1}{2}$ iff some $i \in F$ has $k \in S_i$.*

Lean: `EBaMlProofs.LocalExplanations.Hardness.SetCover.half_lt_sum_iff`, in `proofs/EBaMlProofs/LocalExplanations/Hardness.lean`.

The sum is the number of sets in F that hold k. This number is an integer, so it is greater than $\frac{1}{2}$ iff it is at least 1, that is, iff some set in F holds k.

Theorem 8. (the reduction). *A set $F \subseteq I$ is sufficient for the problem of the reduction iff F is a cover. Hence a cover of at most K sets exists iff a sufficient set of at most K features exists.*

Lean: `EBaMlProofs.LocalExplanations.Hardness.SetCover.sufficient_iff_cover`, `EBaMlProofs.LocalExplanations.Hardness.SetCover.min_size_iff`, in `proofs/EBaMlProofs/LocalExplanations/Hardness.lean`.

ASSUME: an instance of Set Cover and a set F. **PROVE:** F is sufficient iff F is a cover.
⟨1⟩1 F is sufficient iff $-\frac{1}{2} + \sum_{i \in F} h_{k,i} > 0$ for each $k \in U$. This is the definition of Section 5 with $W_k = -\frac{1}{2}$.
⟨1⟩2 For each $k \in U$, $-\frac{1}{2} + \sum_{i \in F} h_{k,i} > 0$ iff $\sum_{i \in F} h_{k,i} > \frac{1}{2}$.
⟨1⟩3 For each $k \in U$, this holds iff some $i \in F$ has $k \in S_i$, by Lemma 13.
⟨1⟩4 **Q.E.D.** The three steps say that F is sufficient iff each element lies in a set of F. The second statement of the theorem follows, because the same F serves both questions and $|F|$ is the same.

The second statement of Theorem 8 is the claim of a reduction. It maps a question about covers to a question about sufficient sets, and the answers are equal.

Corollary 4. (NP-completeness). *The question "does a sufficient set of at most K features exist" for a problem with rational numbers is NP-complete.*

The question is in NP: a set F of features is a certificate, and a check of the sums takes a time that is polynomial in the size of the problem. The question is NP-hard: Set Cover is NP-hard [4], and Theorem 8 reduces it to this question in linear time. Lean checks Theorem 8. The membership in NP and the time of the reduction are not checked in Lean (Section 10).

Theorem 8 is about an abstract table of gains. The next two statements show that a multiclass classifier of the model of Section 2 produces this table. The first one is about a binary model with two bins for each feature.

Lemma 14. (a model of the shape of the reduction). *Let m be a binary model with $z = 0$, with the intercept $b = -\frac{1}{2}$, and with two bins for each feature, both plausible. Let the instance have the bin that scores $a_i \geq 0$, and let the other bin score 0. If $\sum_i a_i > \frac{1}{2}$, then F is sufficient iff $\sum_{i \in F} a_i > \frac{1}{2}$.*

Lean: `EBaMlProofs.LocalExplanations.Hardness.sufficient_of_shape`, in `proofs/EBaMlProofs/LocalExplanations/Hardness.lean`.

ASSUME: the model of the lemma. **PROVE:** F is sufficient iff $\sum_{i \in F} a_i > \frac{1}{2}$.
⟨1⟩1 The logit is $-\frac{1}{2} + \sum_i a_i > 0$, so $\varepsilon = 1$.
⟨1⟩2 Because $z = 0$ and $\varepsilon = 1$, the guaranteed score $g_{i(v)}$ is the score $s_{i(v)}$.
⟨1⟩3 For each feature i, $\min_{v \in P_i} g_{i(v)} = \min(a_i, 0) = 0$, because $a_i \geq 0$.
⟨1⟩4 Hence $W(\varnothing) = \varepsilon b + \sum_i 0 = -\frac{1}{2}$, and $h_i = g_{i(c_i)} - 0 = a_i$.
⟨1⟩5 **Q.E.D.** Lemma 5 gives that F is sufficient iff $\frac{1}{2} < \sum_{i \in F} a_i$.

Theorem 9. (a multiclass classifier with the problem of the reduction). *Let each element of U lie in some set S_i. Let $M(I)$ be the multiclass classifier with the predicted class w and one rival class j_k for each element $k \in U$. Let it have $z = 0$, the intercepts $b_w = 0$ and $b_{j_k} = \frac{1}{2}$, two plausible bins for each feature with the instance in the bin "yes", $s_{w,i} = 0$, $s_{j_k,i}(\text{yes}) = -h_{k,i}$ and $s_{j_k,i}(\text{no}) = 0$. Then a set F is sufficient for the predicted class iff F is a cover.*

Lean: `EBaMlProofs.LocalExplanations.Hardness.sufficient_reason_iff_cover`, `EBaMlProofs.LocalExplanations.Hardness.multiclass`, in `proofs/EBaMlProofs/LocalExplanations/Hardness.lean`.

ASSUME: an instance in which each element lies in a set, and a set F. **PROVE:** F is sufficient iff F is a cover.
⟨1⟩1 F is sufficient iff F is sufficient for the margin model m_{j_k} of each rival, by Lemma 11.
⟨1⟩2 The model m_{j_k} has $z = 0$, the intercept $b_w - b_{j_k} = -\frac{1}{2}$ and the scores $s_{w,i} - s_{j_k,i}$. These scores are $h_{k,i}$ in the bin "yes" and 0 in the bin "no".

⟨1⟩3 The sum $\sum_i h_{k,i}$ counts the sets that hold k. It is at least 1, because k lies in a set. So the condition $\sum_i a_i > \frac{1}{2}$ of Lemma 14 holds for $a_i = h_{k,i}$.
⟨1⟩4 By Lemma 14, F is sufficient for m_{j_k} iff $\sum_{i \in F} h_{k,i} > \frac{1}{2}$. By Lemma 13, this holds iff some $i \in F$ has $k \in S_i$.
⟨1⟩5 **Q.E.D.** By ⟨1⟩1 and ⟨1⟩4, F is sufficient iff each element lies in a set of F.

If an element lies in no set, no cover exists, and a check in linear time finds this case. So the restriction of Theorem 9 does not change the hardness.

The reduction also keeps the quality of an answer. For a property P of sets of features, the *optimum* is the smallest size of a set with P. A set F is a *ρ-approximate* set if F has P and $|F| \leq \rho$ times the optimum. The properties "sufficient" and "cover" are the two cases of this section.

Theorem 10. (the approximation ratio). *Let each element of U lie in some set S_i. Then the following hold, both for the problem of the reduction and for the multiclass classifier $M(I)$ of Theorem 9.*
1. *A number o is the smallest size of a sufficient set iff o is the smallest size of a cover.*
2. *For each ρ, a set F is a ρ-approximate sufficient set iff F is a ρ-approximate cover.*

Lean: `EBaMlProofs.LocalExplanations.Hardness.SetCover.isOptimum_iff`, `EBaMlProofs.LocalExplanations.Hardness.SetCover.isApprox_iff`, `EBaMlProofs.LocalExplanations.Hardness.multiclass_isOptimum_iff`, `EBaMlProofs.LocalExplanations.Hardness.multiclass_isApprox_iff`, in `proofs/EBaMlProofs/LocalExplanations/Hardness.lean`.

By Theorems 8 and 9, the sufficient sets are exactly the covers. The optimum and the ρ-approximate sets depend only on which sets have the property, so they are the same for both properties.

Theorem 10 says that the reduction is *strict*: it maps each solution to itself, and it keeps both the optimum and the ratio. So each guarantee of an approximation algorithm for one problem is the same guarantee for the other. The next statement uses this fact for the lower bound.

Corollary 5. (the same hardness of approximation as Set Cover). *Let ρ be a function of the number m of margins. Suppose that a polynomial-time algorithm gets a covering problem with m margins that has a sufficient set, and returns a sufficient set with at most $\rho(m)$ times the smallest number of features. Then a polynomial-time algorithm gets a Set Cover instance with m elements and returns a cover with at most $\rho(m)$ times the smallest size. Hence, for each $\alpha > 0$, it is NP-hard to find a sufficient set with at most $(1 - \alpha) \ln m$ times the smallest number of features. This holds for a covering problem with rational numbers, and for a multiclass classifier with one rival class for each element.*

ASSUME: an algorithm A as in the statement. **PROVE:** an algorithm for Set Cover with the same ratio.
⟨1⟩1 Given a Set Cover instance, check in linear time that each element lies in a set. If not, no cover exists, and the algorithm says so.
⟨1⟩2 Otherwise build the problem of the reduction in linear time. It has m margins, and a sufficient set exists by Theorem 8, because the set of all indices is a cover.
⟨1⟩3 Run A and return its set. By Theorem 10, this set is a cover with at most $\rho(m)$ times the smallest size of a cover.
⟨1⟩4 **Q.E.D.** For the second sentence, take $\rho(m) = (1 - \alpha) \ln m$. By Corollary 1.5 of Dinur and Steurer [3], for each $\alpha > 0$ it is NP-hard to approximate Set Cover within $(1 - \alpha) \ln n$ for an instance of size n. We read n as the number of elements, as the proof of that corollary does. So an algorithm A with this ratio would give a polynomial-time algorithm for an NP-hard problem. The last sentence follows from Theorem 9 in the same way.

Lean checks Theorem 10, and the paper cites Corollary 1.5 of [3] as stated. The proof of Corollary 5 uses that corollary and the linear time of the reduction. Lean checks neither of them (Section 10).

The upper bound. Set Cover has a polynomial-time algorithm with the ratio $1 + \ln m$ [5]. For the covering problem with rational numbers and one fixed bin for each feature, the problem is a covering integer program with the bounds $x_i \leq 1$, after one change: for rational numbers with the common denominator q, the condition $W_k + \sum_{i \in F} h_{k,i} > 0$ is the same as $W_k + \sum_{i \in F} h_{k,i} \geq \frac{1}{q}$. Kolliopoulos and Young [6] give a

polynomial-time algorithm with the ratio $O(\log m)$ for such programs. So, with the bins fixed, the covering problem and Set Cover have the same approximability up to a constant factor. This paper does not prove the guarantee of Kolliopoulos and Young, and the code does not implement it. The solver `GreedyCover` follows the greedy of Chvátal with the coverage of Wolsey [7] for real gains, but EBaMl proves no ratio for it (Section 9). The counterfactual of a chosen class picks at most one bin for each feature. This constraint is not a covering constraint, and the paper states nothing about its approximability.

Meaning for the code. `MipCover` runs a MIP solver with a time limit, `GreedyCover` and `LocalSearchCover` prove no ratio, and the status `minimality` needs a proof of the MIP solver or the lower bound of Lemma 17. These facts match the hardness. The test `test_the_mip_solves_set_cover_through_the_reduction_of_the_proof` runs the reduction on small instances, and compares the size of `MipCover`, with each of the three solvers of OR-Tools, with the size of a smallest cover.

Correspondence of symbols to the code for this section.

Symbol	Computed by	Function
W_k	`[-0.5] * ELEMENT_COUNT`	`set_cover_problem` of the tests
$h_{k,i}$	`1.0 if element in members else 0.0`	`set_cover_problem` of the tests
the smallest sufficient set	`solver.solve(set_cover_problem(sets), seed=0)`	`MipCover.solve`
a cover of the smallest size	`min(size for size ... if set().union(*chosen) >= set(elements))`	the test of the reduction

9 Solvers, the greedy and the lower bound

This section states what the solvers of `covering_solvers.py` prove. Lean checks none of its statements. The proofs are on paper only, and Section 10 lists this fact.

The problem of the code. `CoveringProblem` extends the problem of Section 5. The features are *groups*, and each group has a finite list of *alternatives* a with a gain $h_{k,a}$ for each margin k. A gain can be negative. A *set of picks* takes at most one alternative from each group, and its size is the number of picks. It is sufficient iff $W_k + \sum_{a \text{ picked}} h_{k,a} > 0$ for each k. The sufficient reason has one alternative for each group, with the gains of `gain_table`, which are not negative. This is the problem of Section 5. The counterfactual of a chosen class j has one alternative for each plausible bin v other than the bin of the instance, with the gain $s_{j,i}(v) - z\sigma_{j,i}(v) - s_{j,i}(c_i)$ for each margin of j, as `_alternatives` computes it.

Lemma 15. (an alternative with no positive gain). *Let the alternative a have $h_{k,a} \leq 0$ for each margin k. If a sufficient set of picks contains a, then the set without a is sufficient and has one pick fewer.*

Each margin of the set without a is the margin of the set plus $-h_{k,a} \geq 0$, so it stays positive.

Lemma 16. (a dominated alternative). *Let a and b be two alternatives of one group with $h_{k,a} \leq h_{k,b}$ for each margin k. If a sufficient set of picks contains a, then the set with b in place of a is sufficient and has the same size.*

The set picks at most one alternative of the group, so it does not contain b, and the swap keeps one pick in the group. Each margin changes by $h_{k,b} - h_{k,a} \geq 0$, so it stays positive.

Corollary 6. (the reduction of the code). *Let `reduce_problem` remove each alternative with no positive gain, and each alternative a that another alternative b of its group dominates, where b has a larger gain for some margin or b comes first. Then the smallest size of a sufficient set is the same before and after the removal, and a sufficient set exists before iff one exists after.*

ASSUME: a sufficient set F of the problem. **PROVE:** a sufficient set of the reduced problem with at most $|F|$ picks. The reduced problem has fewer alternatives, so its sets are sets of the problem, and the other direction is clear.

⟨1⟩1 Remove from F each alternative with no positive gain. The set stays sufficient, and it does not grow, by Lemma 15.
⟨1⟩2 The relation "b dominates a" of the statement is a strict order on the useful alternatives of a group, so each removed alternative a has a dominating alternative that is not removed: a maximal one above a. Swap each such a for it. The set stays sufficient, and its size stays, by Lemma 16.
⟨1⟩3 **Q.E.D.** The result uses only the alternatives that stay.

The reduction does not compare alternatives of different groups. Two groups may be needed together: with one margin, $W = -\frac{3}{2}$ and two groups with the gain 1 each, both groups are needed, and the removal of one of them leaves no sufficient set. So the rule "remove a feature that another feature dominates" of set cover does not hold for this problem. The test `test_the_reduction_keeps_two_groups_with_equal_gains` is this example.

Lemma 17. (the lower bound of the LP). *Let L be the optimum of the linear program that minimizes $\sum_a x_a$ over $0 \le x_a \le 1$, with $\sum_{a \in g} x_a \le 1$ for each group g and $\sum_a h_{k,a} x_a \ge -W_k$ for each margin k. Then each sufficient set of picks has at least $\lceil L \rceil$ picks.*

Let F be a sufficient set of picks, and let $x_a = 1$ for $a \in F$ and $x_a = 0$ otherwise. Then x meets each constraint, because F takes at most one alternative of each group and $W_k + \sum_{a \in F} h_{k,a} > 0 \ge 0$. Hence $|F| = \sum_a x_a \ge L$, and $|F| \ge \lceil L \rceil$ because $|F|$ is an integer.

The code writes the LP of Lemma 17 as a `CoveringModel` (`build_model`), after a division of each margin by the largest absolute value of its numbers, which does not change the set of solutions, and a `MipBackend` solves it. The backend of OR-Tools uses GLOP. GLOP returns L up to its tolerances, so the code subtracts 10^{-6} before the rounding up (`_lower_bound`). The bound therefore holds up to the tolerances of the LP solver. If the backend proves the LP infeasible, Lemma 17 shows that no sufficient set exists, and the solver returns `None`.

What `MipCover` proves. The MIP of the code asks each margin to be at least ε times the size of its numbers, in place of positive, because a solver works with tolerances. The first ε is 10^{-6}. The code checks each set of the solver in Python with the definition above. If the check fails, it solves again with an ε that is 100 times larger, and it tries three values. When the greedy found a set of size g, the MIP also asks for at most $g - 1$ picks. A solver that ends with the status `OPTIMAL` then proves its set smallest among the sets whose margins reach ε, and a solver that proves this MIP infeasible proves the set of the greedy smallest in the same sense (Theorem 5). A set whose smallest margin is positive but below ε times the size of its numbers can be missed. In that case the status `minimality` can say that a set is the smallest when it is not. Lean does not check the certificate of a solver, and no solver on PyPI gives an exact answer in rational numbers.

The time limit. If the time limit ends before the MIP solver proves the optimum, `MipCover` returns the set that the solver found, which is smaller than the greedy set, or else the greedy set, and `minimality` is `AT_LOWER_BOUND` only if its size equals the bound of Lemma 17. If no set is known, the problem of the sufficient reason returns all its features, which Lemma 6 shows to be sufficient when any set is, and the counterfactual of a chosen class raises `SearchLimitError`, because `None` would claim that no set exists. `GreedyCover` and `LocalSearchCover` raise it too when the greedy finds no set of a problem with negative gains.

The greedy. Let $d_k = \varepsilon - W_k$ be the deficit of the margin k and $c_{k(F)}$ the sum of the positive parts of the gains of F for k. The coverage of F is $\sum_{k:d_k>0} \min\left(1, c_{k(F)}/d_k\right)$. Each step adds the alternative of an unused group with the largest gain of coverage, and the lowest position on a tie. The search keeps the last gain of each alternative in a heap and computes a gain again only at the top, as Minoux does [8]. The gain of an alternative never grows when the set grows, because each term $\min(h^+, \max(d_k - c_k, 0))/d_k$ falls when c_k grows. So the lazy search picks what the plain search picks, and the test `test_the_lazy_greedy_picks_what_the_plain_greedy_picks` compares them. Wolsey [7] proves a ratio for the greedy of a submodular cover function. This paper has not checked his statement against the code, with its removal pass and its ε, and a group with several alternatives is outside his setting. So EBaMl claims no ratio for `GreedyCover`. Its guarantee is the bound of Lemma 17.

The local search. `LocalSearchCover` starts from the greedy set. Each step removes a random share of the current picks, at least one, and runs the greedy from the picks that stay, without the removed ones. It keeps the new set if it is not larger, and it remembers the smallest set. The removal pass of the greedy runs in each step, so Theorem 4 holds for each set of the search. The search is not global and proves nothing about the size, except through Corollary 3 and Lemma 17. It stops early when a set reaches the bound. A fixed seed gives the same set.

Correspondence of symbols to the code for this section.

Symbol	Computed by	Function
$h_{k,a}$ of a chosen class	`view.scores[bin_index] - z * view.stds[bin_index] - view.current`	`_alternatives`
Lemmas 15 and 16	`any(gain > 0 ...)` and `_dominates(...)`	`reduce_problem`
L	`result.objective` of the LP	`_lower_bound`
$\lceil L \rceil$	`math.ceil(result.objective - _ROUNDING_TOLERANCE)`	`_lower_bound`
ε	`MARGIN_EPSILON`, times `_EPSILON_STEP`	`MipCover.solve`
at most $g - 1$ picks	`max_picks=len(opening.greedy) - 1`	`MipCover.solve`, `build_model`
the coverage gain	`min(max(gain, 0.0), max(deficit - margin_covered, 0.0)) / deficit`	`_coverage_gain`
a step of the local search	`greedy_cover(problem, stay, excluded=...)`	`LocalSearchCover.solve`

10 Limits

The proof does not cover the following points.

- **Floating point.** The proof works over the real numbers. The code uses 64-bit floating point. A rounding error can change the order of two nearly equal values, or the sign of a margin that is nearly zero.
- **Ties in `np.argmin`.** When two bins reach the minimum of p_i, `np.argmin` returns the first. The proof allows any bin β_i at the minimum, and the value d_i does not depend on the choice. The size of the answer is the same, but the named bin may differ. The proof does not model which one.
- **Completions.** `sample_completions` samples the unknown features from the training weights. The result is an estimate, so there is nothing exact to prove.
- **`new_output`.** The counterfactual reports the probability of the new class from the mean scores, with no deviation. It is a report and not an optimality claim.
- **The uncertainty model of each claim.** The two claims move different scores by z deviations, and the module docstring names both models. For the sufficient reason, each score moves against the decision: the fixed and the unknown features both use g. For the counterfactual, only the changed features use p. The unchanged features keep their mean scores in M (in `_counterfactual`). The proof covers these two models. A counterfactual that also moves the unchanged features needs a different definition of M, and a new proof of Theorem 2.
- **The tie at the threshold.** A change of a prediction above τ that lands exactly on τ gives the robust margin 0, so the code does not count it as a flip, although the value is then "at most τ". This is the same convention as the logit 0 of a classifier. Every change that the code finds does flip the decision (Lemma 8), and "fewest" counts the changes that cross τ strictly. A point mass at τ has no meaning in floating point.
- **The order of the greedy.** The proof allows any order in both passes of the greedy. It proves a set from which no feature can be removed (Theorem 4). About the size, it proves only Corollary 3 and the bound of Lemma 17.

- **The running time in the hardness result.** Lean checks that the reduction is correct (Theorem 8). It does not check that the reduction runs in polynomial time, that the decision problem is in NP (Corollary 4), or Karp's theorem [4]. The proof of Corollary 4 argues the first two in a few lines, and the third is a citation. Corollary 5 uses Corollary 1.5 of [3] as stated. Lean does not check it, and the paper does not re-prove it. The paper reads the size n in that corollary as the number of elements, which is how its proof counts. A reader who wants another measure of size must check the translation of $\ln n$.
- **The upper bound.** The ratio $O(\log m)$ of Kolliopoulos and Young [6] and the ratio $1 + \ln m$ of Chvátal [5] are citations. The paper proves neither. The code implements a greedy of the kind of Chvátal (`GreedyCover`), and it claims no ratio for it.
- **The number of margins.** The reduction needs one margin for each element, so a classifier needs one rival class for each element. A classifier with few classes has few margins. The paper proves nothing about a fixed number of margins that is larger than one. One margin is the case that sorting solves exactly (Theorem 3).
- **The solvers.** A proved optimum rests on the certificate of HiGHS, SCIP or CP-SAT, which Lean does not check, and on their floating-point tolerances. A set whose smallest margin is positive but below ε times the size of its numbers can be missed, so the status `minimality` can be wrong in that case (Section 9). The lower bound depends on the tolerances of GLOP. Lemmas 15 to 17 and Corollary 6 are proved on paper, and Lean checks none of them.
- **The statement `card_search_space`.** Theorem 5 keeps the bound 2^n of a search by size. No function of the code searches by size, so this part describes the oracle of the tests only.
- **The confidence of a decision with several margins.** Each margin moves by z deviations against its own claim, as in Section 2. The proof covers this definition, which is the code. EBMs store no covariance between the scores of two margins, and the proof does not need one.
- **Ties between classes.** Lemma 10 needs $S_w > S_j$ for every rival. When two classes have the same top score, the code takes the first class, and its margin against the other is 0. The proof does not cover this case, and `points_towards` reads the direction of a contribution wrongly there. A tie of two floating-point scores has no meaning.
- **The class j as target.** The Lean model covers the counterfactual "the predicted class is no longer w". It does not contain a counterfactual that asks for a chosen class j, which needs every margin of j at once. The code solves it as a problem of Section 9 with one alternative for each plausible bin (`_target_counterfactual`), so the search runs over every choice of bins. Lemmas 15 to 17 apply to it. Theorems 4 and 5 model one alternative for each feature with gains that are not negative, so they do not apply. The reduction of Section 8 applies to its gain table by the same argument.
- **The probability of the new class.** `Counterfactual.new_output` of a multiclass classifier is the softmax probability from the mean scores after the change. It is a report and not an optimality claim, as for a binary classifier.
- **The correlation between classes.** Lemma 12 shows an upper bound for the deviation of a margin. The code uses the bound, so the claims are conservative, and the proof does not state how much.
- **Interaction terms.** With interaction terms the score is not a sum of one term per feature, and the greedy rule is not minimal. Take three features a, b and c with bins in $\{0, 1\}$, and the instance $(0, 0, 0)$. Let the score be $f = 1 - 0.6a - 0.6b + u(b, c)$, where $u(b, c) = -2$ for $(b, c) = (0, 1)$ and $u(b, c) = 0$ otherwise. The main effects give the drops 0.6, 0.6 and 0 for a, b and c, and the margin is 1. The greedy rule needs $0.6 + 0.6 > 1$, so it returns two features. The change of c alone to 1 gives $f = -1 < 0$ and flips the prediction. The code rejects such models (`read_features`).
- **The empirical link to the code.** The test `test_the_facts_match_the_oracle_of_the_paper` in `tests/unit/ebaml/local/test_facts.py` compares the code with an oracle on random small models of each kind. The oracle enumerates the changes and the completions by the definitions of Section 2, with $z > 0$ and with bins outside P_i. The test checks the link between the code and this model on some inputs. It is not a proof.

11 Computer check

The Lean 4 library in `proofs/` states each lemma, theorem and corollary of Section 3, Section 4, Section 5, Section 6, Section 7 and Section 8 and proves it with the Mathlib library [9]. The file `EBaMlProofs/LocalExplanations/Greedy.lean` holds the abstract part, the file `EBaMlProofs/LocalExplanations/Model.lean` holds the model and the two theorems, the file `EBaMlProofs/LocalExplanations/Covering.lean` holds the problem of several margins, the file `EBaMlProofs/LocalExplanations/Regressor.lean` holds the shift and the answer "inside", and the file `EBaMlProofs/LocalExplanations/Multiclass.lean` holds the models of the rivals, and the file `EBaMlProofs/LocalExplanations/Hardness.lean` holds the reduction from Set Cover. The file `EBaMlProofs/LocalExplanations/Axioms.lean` prints the axioms of the main theorems. The proof has no `sorry`, no `admit` and no new `axiom`. The statements of Section 9 have no Lean file.

A continuous integration job runs `lake build` when the Lean files or the pins change. A reader can also run the check by hand. Section A gives the exact commands, the versions, the expected output and a table from each statement to its Lean name.

12 References

[1] P. Barceló, M. Monet, J. Pérez, and B. Subercaseaux, "Model Interpretability through the Lens of Computational Complexity." [Online]. Available: https://arxiv.org/abs/2010.12265

[2] L. de Moura and S. Ullrich, "The Lean 4 Theorem Prover and Programming Language." [Online]. Available: https://lean-lang.org/

[3] I. Dinur and D. Steurer, "Analytical Approach to Parallel Repetition," in *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, 2014, pp. 624–633.

[4] R. M. Karp, "Reducibility among Combinatorial Problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds., Plenum Press, 1972, pp. 85–103.

[5] V. Chvátal, "A Greedy Heuristic for the Set-Covering Problem," *Mathematics of Operations Research*, vol. 4, no. 3, pp. 233–235, 1979.

[6] S. G. Kolliopoulos and N. E. Young, "Approximation Algorithms for Covering/Packing Integer Programs," *Journal of Computer and System Sciences*, vol. 71, no. 4, pp. 495–505, 2005, [Online]. Available: https://arxiv.org/abs/cs/0205030

[7] L. A. Wolsey, "An Analysis of the Greedy Algorithm for the Submodular Set Covering Problem," *Combinatorica*, vol. 2, no. 4, pp. 385–393, 1982.

[8] M. Minoux, "Accelerated Greedy Algorithms for Maximizing Submodular Set Functions," 1978.

[9] The mathlib Community, "The Lean Mathematical Library." [Online]. Available: https://arxiv.org/abs/1910.09336

A Appendix: how to repeat the check

Versions.

Item	Value
Repository	`https://github.com/johnny-godoy/EBaMl`
Lean sources checked at commit	`4784b6b006e3c9fc629af0c482b5bfb8fac0e453`
Lean	`leanprover/lean4:v4.35.0` (commit `eb6cafb7290e`)
Mathlib	tag `v4.35.0` (commit `56a005855ea17e0b0958f73ec801ae140b0a1679`)
Typst	`0.15.1`

The files in `proofs/EBaMlProofs/`, `proofs/lakefile.toml`, `proofs/lean-toolchain` and `proofs/lake-manifest.json` are the same at the commit above and at the commit of this paper. The functions of Section 2 are in `src/ebaml/local/facts.py`, `fact_features.py`, `counterfactuals.py`, `sufficient_reasons.py` and `completions.py`. Read the paper together with the version of those files at the commit of this paper.

Commands from a clean clone.

```
# install elan, then restart the shell
curl https://elan.lean-lang.org/elan-init.sh -sSf | sh
git clone https://github.com/johnny-godoy/EBaMl && cd EBaMl
git checkout 4784b6b006e3c9fc629af0c482b5bfb8fac0e453
cd proofs
lake exe cache get        # download the compiled Mathlib
lake build                # check every proof
typst compile --root papers papers/local-explanations/paper.typ
```

If `lake exe cache get` cannot reach its server, `lake build` compiles the imported part of Mathlib from source. This takes tens of minutes, and the result is the same.

Expected output of `lake build`. The build ends with `Build completed successfully` and no line that starts with `error`. The file `Axioms.lean` prints these lines, in this order:

```
info: EBaMlProofs/LocalExplanations/Axioms.lean:12:0:
'EBaMlProofs.LocalExplanations.Greedy.threshold_minimal' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:13:0:
'EBaMlProofs.LocalExplanations.Model.min_change' depends on axioms: [propext,
Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:14:0:
'EBaMlProofs.LocalExplanations.Model.min_change_none' depends on axioms: [propext,
Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:15:0:
'EBaMlProofs.LocalExplanations.Model.chosen_bin_differs' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:16:0:
'EBaMlProofs.LocalExplanations.Model.min_sufficient_reason' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:17:0:
'EBaMlProofs.LocalExplanations.Model.min_sufficient_reason_none' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:18:0:
'EBaMlProofs.LocalExplanations.Model.shift_eps' depends on axioms: [propext,
Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:19:0:
```

```
'EBaMlProofs.LocalExplanations.Model.shift_margin_eq' depends on axioms: [propext,
Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:20:0:
'EBaMlProofs.LocalExplanations.Model.shift_robustMargin_neg_iff' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:21:0:
'EBaMlProofs.LocalExplanations.Model.canFlipInside_iff' depends on axioms: [propext,
Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:22:0:
'EBaMlProofs.LocalExplanations.Model.min_change_inside' depends on axioms: [propext,
Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:23:0:
'EBaMlProofs.LocalExplanations.Model.min_change_inside_none' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:24:0:
'EBaMlProofs.LocalExplanations.problemOfModels_sufficient' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:25:0:
'EBaMlProofs.LocalExplanations.Covering.Problem.sufficient_mono' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:26:0:
'EBaMlProofs.LocalExplanations.Covering.Problem.prune_minimal' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:27:0:
'EBaMlProofs.LocalExplanations.Covering.Problem.exact_minimum' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:28:0:
'EBaMlProofs.LocalExplanations.Covering.Problem.card_search_space' depends on
axioms: [propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:29:0:
'EBaMlProofs.LocalExplanations.Covering.Problem.minimum_of_card_le_one' depends on
axioms: [propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:30:0:
'EBaMlProofs.LocalExplanations.Multiclass.logit_marginModel' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:31:0:
'EBaMlProofs.LocalExplanations.Multiclass.winner_iff' depends on axioms: [propext,
Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:32:0:
'EBaMlProofs.LocalExplanations.Multiclass.marginModel_eps' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:33:0:
'EBaMlProofs.LocalExplanations.Multiclass.canFlipClass_iff' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:34:0:
'EBaMlProofs.LocalExplanations.Multiclass.min_change_class' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:35:0:
'EBaMlProofs.LocalExplanations.Multiclass.min_change_class_none' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:36:0:
'EBaMlProofs.LocalExplanations.Multiclass.sufficient_class_iff' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:37:0:
'EBaMlProofs.LocalExplanations.sd_sum_bound' depends on axioms: [propext,
Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:39:0:
```

```
'EBaMlProofs.LocalExplanations.Hardness.SetCover.half_lt_sum_iff' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:40:0:
'EBaMlProofs.LocalExplanations.Hardness.SetCover.sufficient_iff_cover' depends on
axioms: [propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:41:0:
'EBaMlProofs.LocalExplanations.Hardness.SetCover.min_size_iff' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:42:0:
'EBaMlProofs.LocalExplanations.Hardness.sufficient_of_shape' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:43:0:
'EBaMlProofs.LocalExplanations.Hardness.sufficient_reason_iff_cover' depends on
axioms: [propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:44:0:
'EBaMlProofs.LocalExplanations.Hardness.SetCover.isOptimum_iff' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:45:0:
'EBaMlProofs.LocalExplanations.Hardness.SetCover.isApprox_iff' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:46:0:
'EBaMlProofs.LocalExplanations.Hardness.multiclass_isOptimum_iff' depends on axioms:
[propext, Classical.choice, Quot.sound]
info: EBaMlProofs/LocalExplanations/Axioms.lean:47:0:
'EBaMlProofs.LocalExplanations.Hardness.multiclass_isApprox_iff' depends on axioms:
[propext, Classical.choice, Quot.sound]
```

The build may print warnings that Mathlib modules are "designed for use with the module system". These warnings are expected, and they do not affect the result. The output of each `#print axioms` shows only `propext`, `Classical.choice` and `Quot.sound`. The line breaks of long lines may differ. The word `sorryAx` must not appear. To search for it, run `lake build 2>&1 | grep -c sorry`. The expected result is `0`.

The last command writes `papers/local-explanations/paper.pdf` and prints nothing.

Statements and Lean names. The directory is `proofs/EBaMlProofs/LocalExplanations/`. In the table, each prefix stands for a Lean namespace:

- `Greedy.` stands for `EBaMlProofs.LocalExplanations.Greedy.`
- `Model.` stands for `EBaMlProofs.LocalExplanations.Model.`
- `Covering.` stands for `EBaMlProofs.LocalExplanations.Covering.Problem.`
- `MC.` stands for `EBaMlProofs.LocalExplanations.Multiclass.`
- `LE.` stands for `EBaMlProofs.LocalExplanations.`
- `HD.` stands for `EBaMlProofs.LocalExplanations.Hardness.`

Statement	Lean name	File
Definition 1 (top set)	`Greedy.IsTopSet`	`Greedy.lean`
Lemma 1 (top-set bound)	`Greedy.topSet_sum_le`	`Greedy.lean`
Lemma 2 (prefixes)	`Greedy.isTopSet_prefixSet`, `Greedy.card_prefixSet`, `Greedy.sum_prefixSet`	`Greedy.lean`
Theorem 1 (greedy threshold)	`Greedy.threshold_minimal`	`Greedy.lean`
Corollary 1 (no answer)	`Greedy.no_answer`	`Greedy.lean`
Corollary 2 (positive values)	`Greedy.pos_of_threshold`	`Greedy.lean`
Lemma 3 (margin)	`Model.margin_nonneg`	`Model.lean`

Lemma 4 (best change)	`Model.robustMargin_eq`, `Model.robustMargin_ge`, `Model.robustMargin_beta`, `Model.canFlip_iff`	`Model.lean`
Lemma 5 (gain)	`Model.worst_eq`, `Model.sufficient_iff`	`Model.lean`
Theorem 2 (a, b)	`Model.min_change`	`Model.lean`
Theorem 2 (c)	`Model.chosen_bin_differs`	`Model.lean`
Theorem 2 (d)	`Model.min_change_none`	`Model.lean`
Theorem 3 (a, b)	`Model.min_sufficient_reason`	`Model.lean`
Theorem 3 (c)	`Model.min_sufficient_reason_none`	`Model.lean`
Lemma 6 (monotone)	`Covering.sufficient_mono`	`Covering.lean`
Theorem 4 (removal pass)	`Covering.prune_minimal`, `Covering.prune_invariant`	`Covering.lean`
Theorem 5 (exact search)	`Covering.exact_minimum`, `Covering.card_search_space`	`Covering.lean`
Corollary 3 (at most one feature)	`Covering.minimum_of_card_le_one`	`Covering.lean`
Lemma 7 (shift)	`Model.shift_logit`, `Model.shift_eps`, `Model.shift_margin_eq`	`Regressor.lean`
Lemma 8 (crossing)	`Model.shift_robustMargin_neg_iff`	`Regressor.lean`
Theorem 6 (a)	`Model.inside_eps`	`Regressor.lean`
Theorem 6 (b)	`Model.canFlipInside_iff`	`Regressor.lean`
Theorem 6 (c)	`Model.min_change_inside`	`Regressor.lean`
Theorem 6 (d)	`Model.min_change_inside_none`	`Regressor.lean`
Lemma 9 (several margins)	`LE.problemOfModels_sufficient`	`Regressor.lean`
Lemma 10 (the winner)	`MC.logit_marginModel`, `MC.winner_iff`, `MC.marginModel_eps`	`Multiclass.lean`
Theorem 7 (a)	`MC.canFlipClass_iff`	`Multiclass.lean`
Theorem 7 (b)	`MC.min_change_class`	`Multiclass.lean`
Theorem 7 (c)	`MC.min_change_class_none`	`Multiclass.lean`
Lemma 11 (several rivals)	`MC.sufficient_class_iff`	`Multiclass.lean`
Lemma 12 (deviation)	`LE.sd_sum_bound`	`Multiclass.lean`
Lemma 13 (counting the gains)	`HD.SetCover.half_lt_sum_iff`	`Hardness.lean`
Theorem 8 (the reduction)	`HD.SetCover.sufficient_iff_cover`, `HD.SetCover.min_size_iff`	`Hardness.lean`
Corollary 4 (NP-completeness)	Not checked in Lean (Section 10)	none
Lemma 14 (the shape)	`HD.sufficient_of_shape`	`Hardness.lean`
Theorem 9 (multiclass)	`HD.multiclass`, `HD.sufficient_reason_iff_cover`	`Hardness.lean`
Theorem 10 (approximation ratio)	`HD.SetCover.isOptimum_iff`, `HD.SetCover.isApprox_iff`, `HD.multiclass_isOptimum_iff`, `HD.multiclass_isApprox_iff`	`Hardness.lean`
Corollary 5 (same hardness)	Not checked in Lean (Section 10)	none

Lemma 15 (no positive gain)	Not checked in Lean (Section 9)	none
Lemma 16 (a dominated alternative)	Not checked in Lean (Section 9)	none
Corollary 6 (the reduction of the code)	Not checked in Lean (Section 9)	none
Lemma 17 (the lower bound of the LP)	Not checked in Lean (Section 9)	none