No-free-lunch theorem for machine learning

Michikazu Hirata

September 1, 2025

Abstract

This entry is a formalization of the no-free-lunch theorem for machine learning following Section 5.1 of the book *Understanding Machine Learning: From Theory to Algorithms* [1] by Shai Shalev-Shwartz and Shai Ben-David. The theorem states that for binary classification prediction tasks, there is no universal learner, meaning that for every learning algorithms, there exists a distribution on which it fails.

Contents

1 No-Free-Lunch Theorem for ML

theory *No-Free-Lunch-ML*
imports
  HOL−Probability.Probability
begin

1.1 Preliminaries

lemma *sum-le-card-Max-of-nat*:*finite A*
$\Longrightarrow$ *sum f A* $\leq$ (*of-nat* :: - $\Rightarrow$ - ::{*semiring-1*,*ordered-comm-monoid-add*}) (*card A*) $*$ *Max* (*f* ‘ *A*)
  using *sum-bounded-above*[*of A f Max* (*f* ‘ *A*)] **by** *simp*

lemma *card-Min-le-sum-of-nat*: *finite A*
$\Longrightarrow$ (*of-nat* :: - $\Rightarrow$ - ::{*semiring-1*,*ordered-comm-monoid-add*}) (*card A*) $*$ *Min* (*f* ‘ *A*) $\leq$ *sum f A*
  using *sum-bounded-below*[*of A Min* (*f* ‘ *A*) *f*] **by** *simp*

The following lemma is used to show the last equation of the proof of the no-free-lunch theorem in the book [1].

Let A be a finite set. If A is divided into the pairs $(x_1, y_1), \ldots, (x_n, y_n)$ such that $f(x_i) + f(y_i) = k$ for all $i = 1, \ldots, n$. Then, we have $\sum_{x \in A} f(x) = k * |A|/2$.

```
lemma sum-of-const-pairs:
  fixes k :: real
  assumes A:finite A
    and fst ‘ B ∪ snd ‘ B = A fst ‘ B ∩ snd ‘ B = {}
    and inj-on fst B inj-on snd B
    and sum: ⋀x y. (x,y) ∈ B ⟹ f x + f y = k
  shows (∑ x∈A. f x) = k ∗ real (card A) / 2
  using assms
proof(induction A arbitrary: B rule: finite-psubset-induct)
  case ih:(psubset A)
  show ?case
  proof(cases A ={})
    assume A ≠ {}
    then obtain x where x:x ∈ A
      by blast
    then obtain y where xy:(x,y) ∈ B ∨ (y,x) ∈ B
      using ih(3) by fastforce
    then have xy':x ≠ y
      by (metis emptyE fst-eqD ih(4) imageI mem-simps(4) snd-eqD)
    have y:y ∈ A
      using ih(3) xy by force
    have ∗:(∑ a∈A − {x,y}. f a) = k ∗ real (card (A − {x,y})) / 2
    proof −
      consider (x,y) ∈ B | (y,x) ∈ B
        using xy by blast
      then show ?thesis
      proof cases
        assume xy:(x,y) ∈ B
        show ?thesis
        proof(intro ih(2))
          have ∗:fst ‘ (B − {(x, y)}) = fst ‘ B − {x}
            by(subst inj-on-image-set-diff[of fst B]) (use ih(5) xy in auto)
          have ∗∗: snd ‘ (B − {(x, y)}) = snd ‘ B − {y}
            by(subst inj-on-image-set-diff[of snd B]) (use ih(6) xy in auto)
          have x ∉ snd ‘ B y ∉ fst ‘ B
            using ih(4) xy by(force simp: disjoint-iff)+
          thus fst ‘ (B − {(x,y)}) ∪ snd ‘ (B − {(x,y)}) = A − {x,y}
            using ih(3) by(auto simp: ∗ ∗∗)
        qed(use x ih(4) in auto intro!: inj-on-diff ih(5,6,7))
      next
        assume xy:(y,x) ∈ B
        show ?thesis
        proof(intro ih(2))
          have ∗:fst ‘ (B − {(y, x)}) = fst ‘ B − {y}
            by(subst inj-on-image-set-diff[of fst B]) (use ih(5) xy in auto)
          have ∗∗: snd ‘ (B − {(y, x)}) = snd ‘ B − {x}
```

```
        by(subst inj-on-image-set-diff[of snd B]) (use ih(6) xy in auto)
      have y ∉ snd ‘ B x ∉ fst ‘ B
        using ih(4) xy by(force simp: disjoint-iff)+
      thus fst ‘ (B − {(y,x)}) ∪ snd ‘ (B − {(y,x)}) = A − {x,y}
        using ih(3) by(auto simp: ∗ ∗∗)
    qed(use x ih(4) in auto intro!: inj-on-diff ih(5,6,7))
   qed
  qed
  have (∑ a∈A. f a) = (∑ a∈A − {x,y}. f a) + (f x + f y)
   using x y xy′ by (simp add: ih(1) sum-diff)
  also have ... = k ∗ real (card (A − {x,y})) / 2 + (f x + f y)
   by(simp add: ∗)
  also have ... = k ∗ real (card (A − {x,y})) / 2 + k
   using xy ih(7) by fastforce
  also have ... = k ∗ real (card A) / 2
   using x y xy′ by(subst card-Diff-subset)
  (auto simp: of-nat-diff-if card-le-Suc0-iff-eq[OF ih(1)] not-less-eq-eq right-diff-distrib)
  finally show ?thesis .
 qed simp
qed

lemma(in prob-space) Markov-inequality-measure-minus:
 assumes u ∈ borel-measurable M and AE x in M. 0 ≤ u x  AE x in M. 1 ≥ u x
  and [arith]: 0 < (a::real)
 shows 𝒫(x in M. u x > 1 − a) ≥ ((∫ x. u x ∂M) − (1 − a)) / a
proof −
 have [measurable,simp]:integrable M u
  using assms by(auto intro!: integrable-const-bound[where B=1])
 have measure M {x∈space M. u x ≤ 1 − a} = measure M {x∈space M. a ≤ 1
− u x}
  by(rule arg-cong[where f=measure M]) auto
 also have ... ≤ (∫ x. 1 − u x ∂M) / a
  using assms by(intro integral-Markov-inequality-measure) auto
 finally have ∗:measure M {x∈space M. u x ≤ 1 − a} ≤ (∫ x. 1 − u x ∂M) /
a .
 have ((∫ x. u x ∂M) − (1 − a)) / a = 1 − (∫ x. 1 − u x ∂M) / a
  by (auto simp : prob-space diff-divide-distrib)
 also have ... ≤ 1 − measure M {x∈space M. u x ≤ 1 − a}
  using ∗ by simp
 also have ... = measure M {x∈space M. ¬ u x ≤ 1 − a}
  by(intro prob-neg[symmetric]) simp
 also have ... = measure M {x∈space M. u x > 1 − a}
  by(rule arg-cong[where f=measure M]) auto
 finally show ?thesis .
qed
```

1.2 No-Free-Lunch Theorem

In our implementation, a learning algorithm of binary clasification is represented as a function $A : nat \Rightarrow (nat \Rightarrow 'a \times bool) \Rightarrow 'a \Rightarrow bool$ where the first argument is the number of training data, the second argument is the training data ($S\ n = (x_n, y_n)$ denotes the nth data for a training data S), and $A\ m\ S$ is a predictor. The first argument, which denotes the number of training data, is normally used to specify the number of loop executions in learning algorithm. In this formalization, we omit the first argument because we do not need the concrete definitions of learning algorithms.

Let X be the domain set. In order to analyze the error of predictors, we assume that each data (x, y) is obtained from a distribution $\mathcal{D}$ on $X \times \mathbb{B}$. The error of a predictor f with respect to $\mathcal{D}$ is defined as follows.

$$\begin{aligned}\mathcal{L}_{\mathcal{D}}(f) &\overset{\text{def}}{=} \underset{(x,y)\sim\mathcal{D}}{\mathrm{P}}(f(x) \neq y)\\ &= \mathcal{D}(\{(x,y) \in X \times \mathbb{B} \mid f(x) \neq y\})\end{aligned}$$

In these settings, the no-free-lunch theorem states that for any learning algorithm A and $m < |X|/2$, there exists a distribution $\mathcal{D}$ on $X \times \mathbb{B}$ and a predictor f such that

- $\mathcal{L}_{\mathcal{D}}(f) = 0$, and
- $\underset{S\sim\mathcal{D}^m}{\mathrm{P}}\left(\mathcal{L}_{\mathcal{D}}(A(S)) > \frac{1}{8}\right) \geq \frac{1}{7}$.

theorem *no-free-lunch-ML*:
  fixes X :: *'a measure* **and** m :: *nat*
    and A :: $(nat \Rightarrow 'a \times bool) \Rightarrow 'a \Rightarrow bool$
  assumes *X1*:*finite* (*space X*) $\Longrightarrow$ *2* $*$ *m* $<$ *card* (*space X*)
    and *X2*[*measurable*]:$\bigwedge x.\ x \in$ *space X* $\Longrightarrow \{x\} \in$ *sets X*
    and *m*[*arith*]:*0* $<$ *m*
    and *A*[*measurable*]: (λ(*s*,*x*). *A s x*) $\in$ (*PiM* {..<*m*} (λi. *X* $\bigotimes_M$ *count-space* (*UNIV* :: *bool set*))) $\bigotimes_M$ *X*
        $\rightarrow_M$ *count-space* (*UNIV* :: *bool set*)
  shows $\exists\mathcal{D}$ :: (*'a* $\times$ *bool*) *measure*. *sets* $\mathcal{D}$ = *sets* (*X* $\bigotimes_M$ *count-space* (*UNIV* :: *bool set*)) $\wedge$
        prob-space $\mathcal{D}$ $\wedge$
      ($\exists f$. $f \in X \rightarrow_M$ *count-space* (*UNIV* :: *bool set*) $\wedge$ $\mathcal{P}$((*x*, *y*) *in* $\mathcal{D}$. *f x* $\neq$ *y*) = *0*) $\wedge$
      $\mathcal{P}$(*s in Pi*$_M$ {..<*m*} (λi. $\mathcal{D}$). $\mathcal{P}$((*x*, *y*) *in* $\mathcal{D}$. *A s x* $\neq$ *y*) > *1* / *8*) $\geq$ *1* / *7*
proof −
  let *?B* = *count-space* (*UNIV* :: *bool set*)
  let *?B'* = *UNIV* :: *bool set*
  let *?L* = $\lambda D\ f$. $\mathcal{P}$((*x*, *y*) *in D*. *f x* = ($\neg$ *y*))

have *XB*[*measurable*]: $xy \in space\ (X \bigotimes_M\ ?B) \Longrightarrow \{xy\} \in sets\ (X \bigotimes_M\ ?B)$ **for** *xy*
by (*auto simp*: *space-pair-measure sets-Pair*)
have $space\ X \neq \{\}$
using *X1* **by** *force*
have $\exists\, C \subseteq space\ X.\ finite\ C \wedge card\ C = 2 * m$
by (*meson X1 infinite-arbitrarily-large obtain-subset-with-card-n order-less-le*)
then obtain *C* **where** *C*: $C \subseteq space\ X$ *finite C card* $C = 2 * m$
by *blast*
have *C-ne*:$C \neq \{\}$
using *C assms* **by** *force*
have *C-sets*[*measurable*]:$C \in sets\ X$
using *C* **by**(*auto intro*!: *sets.countable*[*OF X2 countable-finite*])
have *meas*[*measurable*]:$\{(x, y).\ (x, y) \in space\ (X \bigotimes_M\ ?B) \wedge g\ x = (\neg\ y)\} \in sets\ (X \bigotimes_M\ ?B)$
if *g*[*measurable*]: $g \in X \rightarrow_M\ ?B$ **for** *g*
proof −
have $\{(x, y).\ (x, y) \in space\ (X \bigotimes_M\ ?B) \wedge g\ x = (\neg\ y)\}$
$= (g\ -`\ \{True\} \cap space\ X) \times \{False\} \cup (g\ -`\ \{False\} \cap space\ X) \times \{True\}$
by(*auto simp*: *space-pair-measure*)
also have ... $\in sets\ (X \bigotimes_M\ ?B)$
by *simp*
finally show *?thesis* **.**
qed

define *fn* **where** $fn \equiv from\text{-}nat\text{-}into\ (C \rightarrow_E (UNIV :: bool\ set))$
define *Dn* **where** $Dn \equiv (\lambda n.\ measure\text{-}of\ (space\ (X \bigotimes_M\ ?B))\ (sets\ (X \bigotimes_M\ ?B))$
$(\lambda U.\ real\ (card\ ((SIGMA\ x{:}C.\ \{fn\ n\ x\}) \cap U))\ /\ real\ (card\ C)))$

have *fn-PiE*:$n < card\ (C \rightarrow_E\ ?B') \Longrightarrow fn\ n \in C \rightarrow_E\ ?B'$ **for** *n*
by (*simp add*: *PiE-eq-empty-iff fn-def from-nat-into*)
have *ex-n*:$f \in C \rightarrow_E\ ?B' \Longrightarrow \exists\, n < card\ (C \rightarrow_E\ ?B').\ f = fn\ n$ **for** *f*
using *bij-betw-from-nat-into-finite*[*OF finite-PiE*[*OF C*(*2*),*of* λ*i*. *?B'*]]
by(*auto simp*: *bij-betw-def fn-def*)
have *fn-inj*: $n < card\ (C \rightarrow_E\ ?B') \Longrightarrow n' < card\ (C \rightarrow_E\ ?B') \Longrightarrow (\bigwedge x.\ x \in C \Longrightarrow fn\ n\ x = fn\ n'\ x) \Longrightarrow n = n'$ **for** $n\ n'$
using *bij-betw-from-nat-into-finite*[*OF finite-PiE*[*OF C*(*2*),*of* λ*i*. *?B'*]] *PiE-ext*[*OF fn-PiE*[*of n*] *fn-PiE*[*of n'*]]
by(*auto simp*: *bij-betw-def fn-def inj-on-def*)

have *fn-meas*[*measurable*]:$fn\ n \in X \rightarrow_M\ ?B$ **for** *n*
proof −
have *countable* $(C \rightarrow_E (UNIV :: bool\ set))$
using *C* **by**(*auto intro*!: *countable-PiE*)
hence $fn\ n \in C \rightarrow_E (UNIV :: bool\ set)$
by (*simp add*: *PiE-eq-empty-iff fn-def from-nat-into*)

```
    hence fn n = (λx. if x ∈ C then fn n x else undefined)
      by auto
    also have ... ∈ X →M ?B
    proof(subst measurable-restrict-space-iff[symmetric])
      have sets (restrict-space X C) = Pow C
      using X2 C by(intro sets-eq-countable) (auto simp: countable-finite sets-restrict-space-iff)
      thus fn n ∈ restrict-space X C →M ?B
        by (simp add: Measurable.pred-def assms(1))
    qed auto
    finally show ?thesis .
  qed

  have sets-Dn[measurable-cong]: ⋀n. sets (Dn n) = sets (X ⨂M ?B)
    and space-Dn:⋀n. space (Dn n) = space (X ⨂M ?B)
    by(simp-all add: Dn-def)
  have emeasure-Dn: emeasure (Dn n) U = ennreal (real (card ((SIGMA x:C. {fn
n x}) ∩ U)) / real (card C))
    (is - = ennreal (?μ U))
    if U[measurable]:U ∈ X ⨂M ?B for U n
  proof(rule emeasure-measure-of[where Ω=space (X ⨂M ?B) and A=sets (X
⨂M ?B)])
    let ?μ' = λU. ennreal (?μ U)
    show countably-additive (sets (Dn n)) ?μ'
      unfolding countably-additive-def
    proof safe
      fix Ui :: nat ⇒ - set
      assume Ui:range Ui ⊆ sets (Dn n) disjoint-family Ui
      have fin: finite {i. (SIGMA x:C. {fn n x}) ∩ Ui i ≠ {}} (is finite ?I)
      proof(rule ccontr)
        assume infinite {i. (SIGMA x:C. {fn n x}) ∩ Ui i ≠ {}}
        with Ui(2)
        have infinite (⋃ ((λi. (SIGMA x:C. {fn n x}) ∩ Ui i) ‘ {i. (SIGMA x:C.
{fn n x}) ∩ Ui i ≠ {}}))
          (is infinite ?U)
            by(intro infinite-disjoint-family-imp-infinite-UNION) (auto simp: dis-
joint-family-on-def)
        moreover have ?U ⊆ (SIGMA x:C. {fn n x})
          by blast
        ultimately have infinite (SIGMA x:C. {fn n x})
          by fastforce
        with C(2) show False
          by blast
      qed
      hence sum:summable (λi. ?μ (Ui i))
        by(intro summable-finite[where N={i. (SIGMA x:C. {fn n x}) ∩ Ui i ≠
{}}]) auto
      have (∑i. ?μ' (Ui i)) = ennreal (∑i. ?μ (Ui i))
        by(intro sum suminf-ennreal2) auto
      also have ... = (∑i∈?I. ?μ (Ui i))
```

by(*subst suminf-finite*[*OF fin*]) *auto*
also have ... = *?*μ' ($\bigcup$ (*range Ui*))
proof −
have ∗:($\sum$ *i*∈*?I*. *real* (*card* ((*SIGMA x*:*C*. {*fn n x*}) ∩ *Ui i*))) = *real* ($\sum$ *i*∈*?I*. (*card* ((*SIGMA x*:*C*. {*fn n x*}) ∩ *Ui i*)))
by *simp*
also have ... = *real* (*card* ($\bigcup$ ((λ*i*. (*SIGMA x*:*C*. {*fn n x*}) ∩ *Ui i*) ‘ *?I*)))
using *C Ui fin* **unfolding** *disjoint-family-on-def*
by(*subst card-UN-disjoint*) *blast*+
also have ... = *real* (*card* ((*SIGMA x*:*C*. {*fn n x*}) ∩ $\bigcup$ (*range Ui*)))
by(*rule arg-cong*[**where** *f*=λ*x*. *real* (*card x*)]) *blast*
finally show *?thesis*
by(*simp add*: *sum-divide-distrib*[*symmetric*])
qed
finally show ($\sum$ *i*. *?*μ' (*Ui i*)) = *?*μ' ($\bigcup$ (*range Ui*)) **.**
qed
qed(*auto simp*: *Dn-def positive-def intro*!:*sets.sets-into-space*)
interpret *Dn*: *prob-space Dn n* **for** *n*
proof
have [*simp*]: (*SIGMA x*:*C*. {*fn n x*}) ∩ *space* (*X* $\bigotimes_M$ *?B*) = (*SIGMA x*:*C*. {*fn n x*})
using *measurable-space*[*OF fn-meas*] *C*(*1*) *space-pair-measure* **by** *blast*
show *emeasure* (*Dn n*) (*space* (*Dn n*)) = *1*
using *C-ne C* **by**(*simp add*: *emeasure-Dn space-Dn*)
qed
interpret *fp*: *finite-product-prob-space* λ*i*. *Dn n* {..<*m*} **for** *n*
by *standard auto*
have *measure-Dn*: *measure* (*Dn n*) *U* = *real* (*card* ((*SIGMA x*:*C*. {*fn n x*}) ∩ *U*)) / *real* (*card C*)
if *U*:*U* ∈ *X* $\bigotimes_M$ *?B* **for** *U n*
using *emeasure-Dn*[*OF U*] **by**(*simp add*: *Dn.emeasure-eq-measure*)
have *measure-Dn'*: *measure* (*Dn n*) *U* = ($\sum$ *x*∈*C*. *of-bool* ((*x*,*fn n x*) ∈ *U*)) / *real* (*card C*)
if *U*[*measurable*]:*U* ∈ *X* $\bigotimes_M$ *?B* **for** *U n*
proof −
have ∗:(*SIGMA x*:*C*. {*fn n x*}) ∩ *U* = (*SIGMA x*:*C*. {*y*. *y* = *fn n x* ∧ (*x*,*y*) ∈ *U*})
by *blast*
have (*x*,*fn n x*) ∈ *U* ⟹ {*y*. *y* = *fn n x* ∧ (*x*, *y*) ∈ *U*} = {*fn n x*}
and (*x*,*fn n x*) ∉ *U* ⟹ {*y*. *y* = *fn n x* ∧ (*x*, *y*) ∈ *U*} = {} **for** *x*
by *blast*+
hence ∗∗:*real* (*card* {*y*. *y* = *fn n x* ∧ (*x*, *y*) ∈ *U*}) = *of-bool* ((*x*,*fn n x*) ∈ *U*) **for** *x*
by *auto*
show *?thesis*
by(*auto simp*: *measure-Dn* ∗ *card-SigmaI*[*OF C*(*2*)] ∗∗)
qed

let *?LossA* = λ*n s*. *?L* (*Dn n*) (*A s*)

```
have [measurable]: (λs. ?LossA n s) ∈ borel-measurable (PiM {..<m} (λi. X
⨂M ?B)) for n
  by measurable (auto simp add: space-Dn)
have Dn-fn-0:𝒫((x, y) in Dn n. fn n x ≠ y) = 0 for n
proof −
  have (SIGMA x:C. {fn n x}) ∩ {(x, y). (x, y) ∈ space (X ⨂M count-space
UNIV) ∧ fn n x = (¬ y)} = {}
    by auto
  thus ?thesis
    by(simp add: measure-Dn space-Dn)
qed

have [measurable]:(SIGMA x:C. {fn n x}) ∈ sets (X ⨂M count-space UNIV)
for n
  by(rule sets.countable) (use C in auto intro!: sets-Pair X2 C(1) countable-finite)
have integ[simp]:integrable (PiM {..<m} (λi. Dn n)) (λs. ?LossA n s) for n
  by(auto intro!: fp.P.integrable-const-bound[where B=1])

have [measurable]:{xn} ∈ sets (PiM {..<m} (λi. X ⨂M ?B))
  and fp-prob:fp.prob n {xn} = 1 / real (card C) ^ m
  if h:xn ∈ {..<m} →E (SIGMA x:C. {fn n x}) for xn n
proof −
  have [simp]: i < m ⟹ xn i ∈ space (X ⨂M ?B) for i
    using h C(1) by(fastforce simp: PiE-def space-pair-measure Pi-def)
  have ∗:{xn} = (ΠE i∈{..<m}. {xn i})
  proof safe
    show ⋀x. x ∈ (ΠE i∈{..<m}. {xn i}) ⟹ x = xn
      by standard (metis PiE-E singletonD h)
  qed(use h in auto)
  also have ... ∈ sets (PiM {..<m} (λi. X ⨂M ?B))
    by measurable
  finally show {xn} ∈ sets (PiM {..<m} (λi. X ⨂M ?B)) .
  have fp.prob n (ΠE i∈{..<m}. {xn i}) = (∏i<m. Dn.prob n {xn i})
    using h by(intro fp.finite-measure-PiM-emb) simp
  also have ... = (1 / real (card C) ^m)
  proof −
    have ⋀i. i < m ⟹ ((SIGMA x:C. {fn n x}) ∩ {xn i}) = {xn i}
      using h by blast
    thus ?thesis
      by(simp add: measure-Dn power-one-over)
  qed
  finally show fp.prob n {xn} = 1 / real (card C) ^ m
    using ∗ by simp
qed


have exp-eq:(∫ s. ?LossA n s ∂(PiM {..<m} (λi. Dn n))) = (∑ s∈{..<m} →E
C. ?LossA n (λi∈{..<m}. (s i, fn n (s i)))) / real (card C) ^ m for n
proof −
```

have (∫ *s. ?LossA n s* ∂(*PiM* {..<*m*} (λ*i. Dn n*)))
= (∫ *s. ?LossA n s* ∗ *indicat-real* (*PiE* {..<*m*} (λ*i.* (*SIGMA x*:*C*. {*fn n x*}))) *s*
+ *?LossA n s* ∗ *indicat-real* (*space* (*PiM* {..<*m*} (λ*i. Dn n*)) − (*PiE* {..<*m*} (λ*i.* (*SIGMA x*:*C*. {*fn n x*})))) *s* ∂(*PiM* {..<*m*} (λ*i. Dn n*)))
by(*auto intro*!: *Bochner-Integration.integral-cong simp*: *indicator-def*)
also have ... = (∫ *s. ?LossA n s* ∗ *indicat-real* (*PiE* {..<*m*} (λ*i.* (*SIGMA x*:*C*. {*fn n x*}))) *s* ∂(*PiM* {..<*m*} (λ*i. Dn n*)))
+ (∫ *s. ?LossA n s* ∗ *indicat-real* (*space* (*PiM* {..<*m*} (λ*i. Dn n*)) − (*PiE* {..<*m*} (λ*i.* (*SIGMA x*:*C*. {*fn n x*})))) *s* ∂(*PiM* {..<*m*} (λ*i. Dn n*)))
by(*rule Bochner-Integration.integral-add*)
(*auto intro*!: *fp.P.integrable-const-bound*[**where** *B*=*1*] *simp*: *mult-le-one*)
also have ... = (∫ *s. ?LossA n s* ∗ *indicat-real* (*PiE* {..<*m*} (λ*i.* (*SIGMA x*:*C*. {*fn n x*}))) *s* ∂(*PiM* {..<*m*} (λ*i. Dn n*)))
proof −
have ∗:(∫ *s. ?LossA n s* ∗ *indicat-real* (*space* (*PiM* {..<*m*} (λ*i. Dn n*)) − (*PiE* {..<*m*} (λ*i.* (*SIGMA x*:*C*. {*fn n x*})))) *s* ∂(*PiM* {..<*m*} (λ*i. Dn n*))) ≥ *0*
by *simp*
have (∫ *s. ?LossA n s* ∗ *indicat-real* (*space* (*PiM* {..<*m*} (λ*i. Dn n*)) − (*PiE* {..<*m*} (λ*i.* (*SIGMA x*:*C*. {*fn n x*})))) *s* ∂(*PiM* {..<*m*} (λ*i. Dn n*)))
≤ (∫ *s. indicat-real* (*space* (*PiM* {..<*m*} (λ*i. Dn n*)) − (*PiE* {..<*m*} (λ*i.* (*SIGMA x*:*C*. {*fn n x*})))) *s* ∂(*PiM* {..<*m*} (λ*i. Dn n*)))
by(*intro integral-mono*) (*auto intro*!: *fp.P.integrable-const-bound*[**where** *B*=*1*] *simp*: *mult-le-one indicator-def*)
also have ... = *1* − *fp.prob n* (*PiE* {..<*m*} (λ*i.* (*SIGMA x*:*C*. {*fn n x*})))
by(*simp add*: *fp.P.prob-compl*)
also have ... = *0*
using *C* **by**(*simp add*: *fp.finite-measure-PiM-emb measure-Dn*)
finally show *?thesis*
using ∗ **by** *simp*
qed
also have ... = (∑ *s*∈{..<*m*} $\to_E$ (*SIGMA x*:*C*. {*fn n x*}). *?LossA n s* ∗ *fp.prob n* {*s*})
using *C* **by**(*auto intro*!: *integral-indicator-finite-real finite-PiE le-neq-trans*)
also have ... = (∑ *s*∈{..<*m*} $\to_E$ (*SIGMA x*:*C*. {*fn n x*}). *?LossA n s*) / *real* (*card C*) ^ *m*
by(*simp add*: *fp-prob sum-divide-distrib*)
also have ... = (∑ *s*∈{..<*m*} $\to_E$ *C*. *?LossA n* (λ*i*∈{..<*m*}. (*s i*, *fn n* (*s i*)))) / *real* (*card C*) ^ *m*
proof −
have ∗:{..<*m*} $\to_E$ (*SIGMA x*:*C*. {*fn n x*}) = (λ*s*. λ*i*∈{..<*m*}. (*s i*, *fn n* (*s i*))) ‘ ({..<*m*} $\to_E$ *C*)
unfolding *set-eq-iff*
proof *safe*
show *s* ∈ {..<*m*} $\to_E$ (*SIGMA x*:*C*. {*fn n x*}) ⟹ *s* ∈ (λ*s*. λ*i*∈{..<*m*}. (*s i*, *fn n* (*s i*))) ‘ ({..<*m*} $\to_E$ *C*) **for** *s*
by(*intro rev-image-eqI*[**where** *b*=*s* **and** *x*=λ*i*∈{..<*m*}. *fst* (*s i*)]) (*force simp*: *PiE-def Pi-def extensional-def*)+
qed *auto*

have ∗∗:*inj-on* (λ*s*. λ*i*∈{..<*m*}. (*s i*, *fn n* (*s i*))) ({..<*m*} $\to_E$ *C*)
by(*intro inj-onI*) (*metis* (*mono-tags*, *lifting*) *PiE-ext prod.simps*(*1*) *restrict-apply*′)
show *?thesis*
by(*subst sum.reindex*[**where** *A*={..<*m*} $\to_E$ *C* **and** *h*=λ*s*. λ*i*∈{..<*m*}. (*s i*, *fn n* (*s i*)),*simplified*,*symmetric*])
(*use* ∗ ∗∗ **in** *auto*)
qed
finally show *?thesis* **.**
qed

have *eqL*:*?L* (*Dn n*) *h* = (∑*x*∈*C*. *of-bool* (*h x* = (¬ *fn n x*))) / *real* (*card C*) **if** *h*[*measurable*]:*h* ∈ *X* $\to_M$ *?B* **for** *n h*
proof −
have *?L* (*Dn n*) *h* = (∑*x*∈*C*. *of-bool* ((*x*, *fn n x*) ∈ *space* (*X* $\bigotimes_M$ *?B*) ∧ *h x* = (¬ *fn n x*))) / *real* (*card C*)
by(*simp add*: *space-Dn measure-Dn*′)
also have ... = (∑*x*∈*C*. *of-bool* (*h x* = (¬ *fn n x*))) / *real* (*card C*)
using *C* **by**(*auto simp*: *space-pair-measure Collect-conj-eq Int-assoc*[*symmetric*])
finally show *?thesis* **.**
qed

have *nz1*[*arith*]:*real* (*card* (*C* $\to_E$ *?B*′)) > *0 real* (*card C*) > *0 0* < *real* (*card* ({..<*m*} $\to_E$ *C*))
using *C*(*2*) *C-ne* **by**(*simp-all add*: *card-funcsetE card-gt-0-iff*)

have *ne*:*finite* ((λ*n*. *fp.expectation n*
(λ*s*. *Dn.prob n* {(*x*, *y*). (*x*, *y*) ∈ *space* (*Dn n*) ∧ *A s x* = (¬ *y*)})) ‘ {..<*card* (*C* $\to_E$ *?B*′)})
((λ*n*. *fp.expectation n*
(λ*s*. *Dn.prob n* {(*x*, *y*). (*x*, *y*) ∈ *space* (*Dn n*) ∧ *A s x* = (¬ *y*)})) ‘ {..<*card* (*C* $\to_E$ *?B*′)}) ≠ {} (**is** *?ne*)
proof −
have *0* < *card* (*C* $\to_E$ *?B*′)
using *C-ne C*(*2*) **by**(*auto simp*: *card-gt-0-iff finite-PiE*)
thus *?ne*
by *blast*
qed *simp*

have *max-geq-q*:(*MAX n*∈{..<*card* (*C* $\to_E$ *?B*′)}. (∫ *s*. *?LossA n s* ∂(*PiM* {..<*m*} (λ*i*. *Dn n*)))) ≥ *1* / *4* (**is** - ≤ *?Max*)
proof −

have (*MIN s*∈{..<*m*} $\to_E$ *C*. (∑ *n*<*card* (*C* $\to_E$ *?B*′). *?LossA n* (λ*i*∈{..<*m*}. (*s i*, *fn n* (*s i*)))) / *real* (*card* (*C* $\to_E$ *?B*′)))
≤ *?Max* (**is** *?Min1* ≤ -)
proof −
have *?Min1*

```
        ≤ (∑ s∈{..<m} →E C.
            (∑ n<card (C →E ?B').
                    ?LossA n (λi∈{..<m}. (s i, fn n (s i)))) / real (card (C →E
?B'))) / real (card ({..<m} →E C))
      proof(subst pos-le-divide-eq)
        show ?Min1 ∗ real (card ({..<m} →E C))
          ≤ (∑ s∈{..<m} →E C. (∑ n<card (C →E ?B'). ?LossA n (λi∈{..<m}.
(s i, fn n (s i)))) / real (card (C →E ?B')))
       using C by(simp add: mult.commute) (auto intro!: finite-PiE card-Min-le-sum-of-nat)
      qed fact
      also have ...
          = (∑ s∈{..<m} →E C.
              (∑ n<card (C →E ?B').
                    ?LossA n (λi∈{..<m}. (s i, fn n (s i)))) / real (card (C →E
?B'))) / real (card C) ^ m
        by(simp add: card-PiE)
      also have ...
            = (∑ n<card (C →E ?B').
                (∑ s∈{..<m} →E C.
                  ?LossA n (λi∈{..<m}. (s i, fn n (s i)))) / real (card C) ^ m) /
real (card (C →E ?B'))
        unfolding sum-divide-distrib[symmetric] by(subst sum.swap) simp
      also have ... ≤ ?Max
      proof −
        have real (card (C →E ?B')) ∗ ?Max
          = real (card (C →E ?B'))
            ∗ (MAX n∈{..<card (C →E ?B')}. (∑ s∈{..<m} →E C. ?LossA n
(λi∈{..<m}. (s i, fn n (s i)))) / real (card C) ^ m)
          by (simp add: exp-eq)
        also have ... ≥ (∑ n<card (C →E ?B'). (∑ s∈{..<m} →E C. ?LossA n
(λi∈{..<m}. (s i, fn n (s i)))) / real (card C) ^ m)
          using sum-le-card-Max-of-nat[of {..<card (C →E ?B')}] finite-PiE[OF
C(2)] by auto
        finally show ?thesis
          by(subst pos-divide-le-eq) (simp, argo)
      qed
      finally show ?thesis .
    qed


    have 1 / 4 ≤ ?Min1
    proof(safe intro!: Min-ge-iff[THEN iffD2])
      fix s
      assume s: s ∈ {..<m} →E C
      hence [measurable]: (λi∈{..<m}. (s i, fn n (s i))) ∈ space (PiM {..<m} (λi.
X ⨂M ?B)) for n
        using C by(auto simp: space-PiM space-pair-measure)
      let ?V = C − (s ‘ {..<m})
      have fin-V:finite ?V
```

```
      using C by blast
    have cardV: card ?V ≥ m
    proof −
      have card (s ‘ {..<m}) ≤ m
        by (metis card-image-le card-lessThan finite-lessThan)
      hence m ≤ card C − card (s ‘ {..<m})
        using C(3) by simp
      also have card C − card (s ‘ {..<m}) ≤ card ?V
        by(rule diff-card-le-card-Diff) simp
      finally show ?thesis .
    qed
    hence V-ne: ?V ≠ {} card ?V > 0
      using m by force+
    have (1 / 2) ∗ (1 / 2)
        = (1 / 2)
        ∗ (MIN v∈?V. (∑n<card (C →E ?B'). of-bool (A (λi∈{..<m}. (s i, fn
n (s i))) v = (¬ fn n v))) / real (card (C →E ?B')))
    proof(rule arg-cong[where f=(∗) (1 / 2)])
      have (∑n<card (C →E ?B'). of-bool (A (λi∈{..<m}. (s i, fn n (s i))) v
= (¬ fn n v))) / real (card (C →E ?B')) = 1 / 2
          if v:v ∈ ?V for v
      proof −
        define B where B ≡ {(n, n')|n n'. n < card (C →E ?B') ∧ fn n v =
False ∧ n' < card (C →E ?B')
                                        ∧ fn n' v = True ∧ (∀x∈C − {v}. fn n x =
fn n' x)}
        have B1:fst ‘ B ∪ snd ‘ B = {..<card (C →E ?B')}
        proof −
          have n ∈ fst ‘ B ∪ snd ‘ B if n:n < card (C →E ?B') for n
          proof(cases fn n v = True)
            assume h:fn n v = True
            let ?fn' = λx. if x = v then False else fn n x
            have fn':⋀x. x ≠ v ⟹ fn n x = ?fn' x ?fn' v = False
              by auto
            hence fn'1:?fn' ∈ C →E ?B'
              using fn-PiE[OF n] v by auto
            then obtain n' where n': n' < card (C →E ?B') fn n' = ?fn'
              using ex-n by (metis (lifting))
            hence (n',n) ∈ B
              using n' fn'1 fn-PiE[OF n] n h fn' by(auto simp: B-def)
            thus ?thesis
              by force
          next
            assume h:fn n v ≠ True
            let ?fn' = λx. if x = v then True else fn n x
            have fn':⋀x. x ≠ v ⟹ fn n x = ?fn' x ?fn' v = True
              by auto
            hence fn'1:?fn' ∈ C →E ?B'
              using fn-PiE[OF n] v by auto
```

then obtain n' **where** n': $n' <$ *card* ($C \rightarrow_E$ *?B'*) *fn* n' = *?fn'*
using *ex-n* **by** (*metis* (*lifting*))
hence $(n,n') \in B$
using n' *fn'1 fn-PiE*[*OF n*] *n h fn'* **by**(*auto simp*: *B-def*)
thus *?thesis*
by *force*
qed
moreover have $\bigwedge n.\ n \in$ *fst* ' $B \cup$ *snd* ' $B \Longrightarrow n <$ *card* ($C \rightarrow_E$ *?B'*)
by(*auto simp*: *B-def*)
ultimately show *?thesis*
by *blast*
qed
have *B2*:*fst* ' $B \cap$ *snd* ' B = {}
by(*auto simp*: *B-def*)
have *B3*: *inj-on fst B*
by(*auto intro*!: *fn-inj inj-onI simp*: *B-def*)
have *B4*: *inj-on snd B*
by(*fastforce intro*!: *fn-inj inj-onI simp*: *B-def*)
have *B5*:*of-bool* (*A* ($\lambda i \in \{..<m\}$. (*s i*, *fn n* (*s i*))) *v*
= ($\neg$ *fn n v*)) + *of-bool* (*A* ($\lambda i \in \{..<m\}$. (*s i*, *fn n'* (*s i*))) *v* = ($\neg$
fn n' v)) = (*1* :: *real*)
if *nn'*:$(n,n') \in B$ **for** *n n'*
proof −
have ($\lambda i \in \{..<m\}$. (*s i*, *fn n* (*s i*))) = ($\lambda i \in \{..<m\}$. (*s i*, *fn n'* (*s i*)))
by *standard* (*use s nn' v* **in** *auto simp*: *B-def*)
thus *?thesis*
using *nn'* **by**(*auto simp*: *B-def*)
qed
have ($\sum n<$*card* ($C \rightarrow_E$ *?B'*). *of-bool* (*A* ($\lambda i \in \{..<m\}$. (*s i*, *fn n* (*s i*))) *v*
= ($\neg$ *fn n v*)))
= *1* ∗ *real* (*card* {..<*card* ($C \rightarrow_E$ *?B'*)}) / *2*
by(*intro sum-of-const-pairs*[**where** *B*=*B*] *B1 B2 B3 B4 B5*) *simp*
thus *?thesis*
by *simp*
qed
thus *1* / *2* = (*MIN* $v \in ?V$. ($\sum n<$*card* ($C \rightarrow_E$ *?B'*). *of-bool* (*A* ($\lambda i \in \{..<m\}$.
(*s i*, *fn n* (*s i*))) *v* = ($\neg$ *fn n v*))) / *real* (*card* ($C \rightarrow_E$ *?B'*)))
by (*metis* (*mono-tags*, *lifting*) *V-ne*(*1*) *fin-V obtains-MIN*)
qed
also have ...
≤ (*1* / *2*)
∗ (($\sum v \in ?V$. ($\sum n<$*card* ($C \rightarrow_E$ *?B'*). *of-bool* (*A* ($\lambda i \in \{..<m\}$. (*s i*, *fn
n* (*s i*))) *v* = ($\neg$ *fn n v*)))
/ *real* (*card* ($C \rightarrow_E$ *?B'*)))
/ *real* (*card ?V*))
using *V-ne* **by**(*intro mult-le-cancel-left-pos*[*THEN iffD2*] *pos-le-divide-eq*[*THEN
iffD2*])
(*simp-all add*: *Groups.mult-ac*(*2*) *card-Min-le-sum-of-nat fin-V*)
also have ...

$= (\sum n<card\ (C \to_E\ ?B').\ ((\sum v\in ?V.\ of\text{-}bool\ (A\ (\lambda i\in\{..<m\}.\ (s\ i,\ fn\ n\ (s\ i)))\ v = (\neg\ fn\ n\ v)))$
$/\ (2 * real\ (card\ ?V)))\ /\ real\ (card\ (C \to_E\ ?B')))$

unfolding *sum-divide-distrib*[*symmetric*] **by**(*subst sum.swap*) *simp*

also have ... $\le (\sum n<card\ (C \to_E\ ?B').\ ?LossA\ n\ (\lambda i\in\{..<m\}.\ (s\ i,\ fn\ n\ (s\ i)))\ /\ real\ (card\ (C \to_E\ ?B')))$

proof(*safe intro*!: *sum-mono divide-right-mono*)

fix *n*

have $(\sum v\in ?V.\ of\text{-}bool\ (A\ (\lambda i\in\{..<m\}.\ (s\ i,\ fn\ n\ (s\ i)))\ v = (\neg\ fn\ n\ v)))\ /\ (2 * real\ (card\ ?V))$
$\le (\sum v\in ?V.\ of\text{-}bool\ (A\ (\lambda i\in\{..<m\}.\ (s\ i,\ fn\ n\ (s\ i)))\ v = (\neg\ fn\ n\ v)))\ /\ real\ (card\ C)$

using *cardV* **by**(*auto simp*: *C*(*3*) *intro*!: *divide-left-mono sum-nonneg*)

also have ... $\le (\sum x\in C.\ of\text{-}bool\ (A\ (\lambda i\in\{..<m\}.\ (s\ i,\ fn\ n\ (s\ i)))\ x = (\neg\ fn\ n\ x)))\ /\ real\ (card\ C)$

using *C* **by**(*intro sum-mono2 divide-right-mono*) *auto*

also have ... $= ?LossA\ n\ (\lambda i\in\{..<m\}.\ (s\ i,\ fn\ n\ (s\ i)))$

by(*simp add*: *eqL*)

finally show $(\sum v\in ?V.\ of\text{-}bool\ (A\ (\lambda i\in\{..<m\}.\ (s\ i,\ fn\ n\ (s\ i)))\ v = (\neg\ fn\ n\ v)))\ /\ (2 * real\ (card\ ?V))$
$\le ?LossA\ n\ (\lambda i\in\{..<m\}.\ (s\ i,\ fn\ n\ (s\ i)))$ **.**

qed *simp*

finally show $1\ /\ 4 \le (\sum n<card\ (C \to_E\ ?B').\ ?LossA\ n\ (\lambda i\in\{..<m\}.\ (s\ i,\ fn\ n\ (s\ i))))\ /\ real\ (card\ (C \to_E\ ?B'))$

by(*simp add*: *sum-divide-distrib*)

qed(*use m C* **in** *auto intro*!: *finite-PiE simp*: *PiE-eq-empty-iff*)

also have ... $\le ?Max$

by *fact*

finally show *?thesis* **.**

qed

hence $\exists n.\ n < card\ (C \to_E\ ?B') \wedge (\int s.\ ?LossA\ n\ s\ \partial(PiM\ \{..<m\}\ (\lambda i.\ Dn\ n))) \ge 1\ /\ 4$

using *Max-ge-iff*[*OF ne*] **by** *blast*

then obtain *n* **where** $n{:}n < card\ (C \to_E\ ?B')\ (\int s.\ ?LossA\ n\ s\ \partial(PiM\ \{..<m\}\ (\lambda i.\ Dn\ n))) \ge 1\ /\ 4$

by *blast*

have $1\ /\ 7 \le ((\int s.\ ?LossA\ n\ s\ \partial(PiM\ \{..<m\}\ (\lambda i.\ Dn\ n))) - (1 - 7/\ 8))\ /\ (7\ /\ 8)$

using *n* **by** *argo*

also have ... $\le \mathcal{P}(s\ in\ Pi_M\ \{..<m\}\ (\lambda i.\ Dn\ n).\ \mathcal{P}((x,\ y)\ in\ Dn\ n.\ A\ s\ x = (\neg\ y)) > 1 - 7\ /\ 8)$

by(*intro fp.Markov-inequality-measure-minus*) *auto*

also have ... $= \mathcal{P}(s\ in\ Pi_M\ \{..<m\}\ (\lambda i.\ Dn\ n).\ \mathcal{P}((x,\ y)\ in\ Dn\ n.\ A\ s\ x = (\neg\ y)) > 1\ /\ 8)$

by *simp*

finally have $1\ /\ 7 \le \mathcal{P}(s\ in\ Pi_M\ \{..<m\}\ (\lambda i.\ Dn\ n).\ \mathcal{P}((x,\ y)\ in\ Dn\ n.\ A\ s\ x = (\neg\ y)) > 1\ /\ 8)$ **.**

thus *?thesis*
using *Dn-fn-0*[*of n*]
by(*auto intro*!: *exI*[**where** *x*=*Dn n*] *exI*[**where** *x*=*fn n*] *simp*: *sets-Dn Dn.prob-space-axioms*)
qed

end

References

[1] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.