Empirical Investigation of an Open Conjecture: AdaBoost Always Cycles? (Global Dynamics Conjecture)

Agentic NL→Lean 4 Pipeline
Job #50

April 26, 2026

Abstract

This report documents the empirical investigation of an open mathematical conjecture that could not be formally proved or disproved in Lean 4 with Mathlib. Numerical experiments were conducted to gather evidence for or against the conjecture. The empirical verdict is: **Inconclusive**. The conjecture remains formally open.

1 Conjecture Statement

Conjecture 1.

```
AdaBoost Always Cycles? (Global Dynamics Conjecture)

Let

=
{
(



,



)
}

=
1

S={(x
i


,y
i


)}
i=1
m
```

```
 be a fixed binary-labeled dataset with


{-

1
,
+
1
}
y
i -


{1,+1}, and let

H be a weak-hypothesis class. Consider the discrete AdaBoost update of
    Freund & Schapire (1997), in the exhaustive weak-learner regime used in
    Rudin et al. (2012), with weight vectors Δ -




1
w
t Δ-


m1
 (the probability simplex). At iteration

t, choose


arg

max


=
1
```

```
,
(
)
,
h
t
arg
hH
max
i=1
m
w
t,i
y
i
h(x
i
),
define weighted error
=
=
1
```

```
,


1
{


(


)



}
,
t


=
i=1

m


w
t,i


1{h
t


(x
i


)

=y
i


},

and update with



=
1
2
log
```

```
1-
,
+
1
,
=
,
exp
(-
(
)
)
,
t
=
2
1
log
t-
1
t
```

```
,w
t+1,i


=
Z
t


w
t,i


exp−(
t


y
i


h
t


(x
i


))


,

where


Z
t


 normalizes to




+
1
,

=
```

1

i

w
t+1,i

=1.

Equivalently, with finite hypothesis set

=
{

~
1
,…

,

~

}
H={
h
~
1…

,,
h
~
N

} and matrix

{−

1
,
+
1
}

×

M−{1,+1}
m×N
 defined by

```
=


~
(

)
M
ij

=y
i

h
~
j

(x
i

), step

t selects



arg

max


[

]
(




)

,
```

```
=

~



.
j
t


arg
j[N]
max


(w
t



M)
j


,h
t


=
h
~
j
t




.

As specified in Rudin et al. (2012), if this argmax is not unique, ties are
    broken in a fixed deterministic way (for concreteness: pick the smallest
    index

j). The generic no-tie condition means the argmax is unique at every iterate
    , i.e.

(




)
```

```
(



)

for all



 and all

,
(w
t



M)
j


=(w
t



M)
j


for all j

=j

 and all t,

equivalently,


w
t


 never lands on a tie boundary between weak-hypothesis regions of the
     simplex.

This induces a discrete dynamical system
```

```
:Δ -


1↠Δ -
```

2 Status

Formal Status: OPEN — no Lean 4 proof or disproof was found.
Empirical Verdict: Inconclusive

The pipeline attempted formal verification in Lean 4 with Mathlib but was unable to produce a compiling proof or disproof. Empirical testing was then conducted to gather numerical evidence.

3 Experiment Code (Basic)

```
import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import random
import time

print("=" * 70)
print("=== EXPERIMENT PLAN ===")
print("=" * 70)
print("""
CONJECTURE: AdaBoost Always Cycles (Rudin et al. 2012 setting)
  Discrete AdaBoost on a fixed dataset with finite weak-hypothesis class
  H induces a deterministic map T on the simplex Delta_{m-1}. With
  deterministic tie-breaking (smallest index) and the generic no-tie
  condition, the conjecture states that the iterates {w_t} ALWAYS
  become eventually periodic.

EQUIVALENT FORMULATION:
  With margin matrix M_{ij} = y_i * h_j(x_i) in {-1,+1}^{m x N}:
    j_t = argmax_j (w_t^T M)_j  (smallest index broken)
    eps_t = sum_i w_{t,i} * 1{M_{i,j_t} = -1}
    alpha_t = 0.5 * log((1-eps_t)/eps_t)
    w_{t+1,i}  w_{t,i} * exp(-alpha_t * M_{i,j_t})

EXPERIMENTAL STRATEGY (multiple angles):
  (A) Random sampling across various (m,N) shapes
  (B) Edge / boundary cases (Hadamard, redundant, anti-symmetric)
  (C) Counterexample search: systematic enumeration for small (m,N)
  (D) Asymptotic / scaling: cycle length vs problem size
  (E) Stability verification via two-period closure + symbol periodicity
""")
```

```
rng = np.random.default_rng(20260425)
random.seed(20260425)
t_start = time.time()

# ---------------------------------------------------------------------
# Core AdaBoost dynamics
# ---------------------------------------------------------------------
def adaboost_step(w, M):
    scores = w @ M
    j = int(np.argmax(scores))          # smallest index argmax
    col = M[:, j]
    eps = float(np.sum(w * (col < 0)))
    if eps <= 1e-14:
        return None, j, eps, "eps_zero"
    if eps >= 1 - 1e-14:
        return None, j, eps, "eps_one"
    alpha = 0.5 * np.log((1.0 - eps) / eps)
    w_new = w * np.exp(-alpha * col)
    s = w_new.sum()
    if s <= 0 or not np.isfinite(s):
        return None, j, eps, "numerical"
    return w_new / s, j, eps, "ok"


def find_period(M, T0=400, T1=600, tol=1e-9):
    m, N = M.shape
    w = np.ones(m) / m
    for t in range(T0):
        w, j, eps, st = adaboost_step(w, M)
        if st != "ok":
            return {"status": "degenerate_" + st, "period": -1,
                    "min_dist": np.nan}
    w0 = w.copy()
    min_dist = np.inf
    period = -1
    for k in range(1, T1 + 1):
        w, j, eps, st = adaboost_step(w, M)
        if st != "ok":
            return {"status": "degenerate_" + st, "period": -1,
                    "min_dist": np.nan}
        d = np.linalg.norm(w - w0)
        if d < min_dist:
            min_dist = d
        if d < tol and period == -1:
            period = k
            break
    if period == -1:
        return {"status": "no_cycle_found", "period": -1,
                "min_dist": min_dist}
    return {"status": "cycle", "period": period, "min_dist": min_dist}


def verify_cycle(M, period, T0=400, tol=1e-8):
```

```
    m, N = M.shape
    w = np.ones(m) / m
    for t in range(T0):
        w, j, eps, st = adaboost_step(w, M)
        if st != "ok":
            return False, "burnin_degenerate"
    w0 = w.copy()
    js1, js2 = [], []
    for k in range(period):
        w, j, eps, st = adaboost_step(w, M)
        js1.append(j)
        if st != "ok":
            return False, "deg"
    d1 = np.linalg.norm(w - w0)
    for k in range(period):
        w, j, eps, st = adaboost_step(w, M)
        js2.append(j)
        if st != "ok":
            return False, "deg"
    d2 = np.linalg.norm(w - w0)
    sym_match = (js1 == js2)
    ok = (d1 < tol) and (d2 < tol) and sym_match
    return ok, {"d1": d1, "d2": d2, "symbols_match": sym_match}

# ------------------------------------------------------------------
# (A) Random sampling experiment
# ------------------------------------------------------------------
print("\n[A] RANDOM SAMPLING EXPERIMENT")
print("-" * 70)

shapes = [(3,4),(4,5),(5,6),(6,8),(8,10),(10,12)]
trials_per_shape = 50
all_records = []

for (m, N) in shapes:
    successes = 0
    periods = []
    degens = 0
    nocyc = 0
    for trial in range(trials_per_shape):
        M = rng.choice([-1, 1], size=(m, N)).astype(float)
        res = find_period(M, T0=300, T1=500, tol=1e-9)
        if res["status"] == "cycle":
            successes += 1
            periods.append(res["period"])
        elif res["status"].startswith("degenerate"):
            degens += 1
        else:
            nocyc += 1
        all_records.append({"m": m, "N": N, **res})
    pavg = np.median(periods) if periods else -1
    pmax = max(periods) if periods else -1
    print(f"  shape (m={m:2d}, N={N:2d}): cycles={successes:3d}/"
```

```
          f"{trials_per_shape},␣degenerate={degens:3d},␣no_cycle={nocyc:3d},
              ␣"
          f"median_period={pavg:.0f},␣max_period={pmax}")

# ----------------------------------------------------------------------
# (B) Edge / boundary cases
# ----------------------------------------------------------------------
print("\n[B]␣EDGE␣/␣BOUNDARY␣CASES")
print("-" * 70)

edge_cases = []
edge
# ... [truncated]
```

4 Experiment Code (Advanced)

```
"""
Advanced numerical experiment for the AdaBoost Cycling Conjecture
(--RudinSchapireDaubechies regime, deterministic smallest-index tie-break).

This goes WELL beyond the basic FP64 experiment by adding:
  - multi-precision (mpmath) orbit recomputation as the analog of grid
    refinement for a discrete dynamical system,
  - exact-loss energy/identity monitoring Π _t Z_t = (1/m) Σ_i exp(-y_i F_T(
     x_i)),
  - long-horizon (T = 4000) certified periodicity at 200-bit precision,
  - parameter sensitivity over (m, N) and tie-break choice,
  - convergence-slope diagnostic  log2(closure_distance) vs. precision.
"""

import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import numpy as np
import mpmath as mp
import math, time, random
from itertools import product

t0 = time.time()
rng = np.random.default_rng(20260425)
random.seed(20260425)

# ======================================================================
# 1.  PLAN
# ======================================================================
print("===␣ADVANCED␣EXPERIMENT␣PLAN␣===")
print("""
TARGET CONJECTURE
   For a fixed +/-1 margin matrix  M   {-1,+1}^{m×N}, the discrete
   AdaBoost map T : Δ^{m-1} →Δ  ^{m-1}
        j_t = argmax_j (w_t^T M)_j        (smallest index on ties),
```

```
       _t = (1 - (w_t^T M)_{j_t})/2,
       _t = 1/2 log ((1-_t) /_t),
       w_{t+1,i}  w_{t,i} exp (-_t M_{i,j_t})
   is conjectured to produce eventually periodic orbits on every
   non-tie initial weight w_0.

WHY THE BASIC EXPERIMENT IS INSUFFICIENT
   The double-precision FP64 sweep produced ~45% "no_cycle_found"
   trajectories, but *cannot distinguish*
     (a) genuinely aperiodic orbits  (= a counterexample),
     (b) cycle period > horizon,
     (c) finite-precision drift past tie-hyperplanes (numerical only).

RESEARCH-GRADE PROTOCOL
   (A) FP64 sweep over (m,N)  {3..8}×{3,4,6,8}; harvest "no_cycle" cases.
   (B) MULTI-PRECISION CONVERGENCE TEST.  Re-run those problem orbits at
       mpmath precisions {53,100,200,400} bits.  Track the minimum
       closure distance d(prec) = min_p ||w_T - w_{-Tp}||_2 .  If the
       conjecture is true,  d(prec) ↓ 0 exponentially in prec.
   (C) EXACT INVARIANT (energy) MONITORING.  AdaBoost satisfies the
       structural identityΠ
            _{t<T} Z_t  =  (1/m) Σ_i exp(-y_i F_T(x_i)),
       with F_T(x_i) = Σ_t  _t M_{i,j_t}.  We verify this in mpmath.
       Drift = a numerical artifact, not dynamics.
   (D) LONG-HORIZON CERTIFICATION.  T = 4000 at prec = 200 bits.
       Cycles must remain exactly periodic (closure < 10^{-50}) and
       the Floquet-style rate  log Φ_t / t  must converge to (log Φ_p)/p.
   (E) TIE-BREAK SENSITIVITY.  Compare smallest-index vs largest-index
       tie-break to detect tie-hyperplane hits.
   (F) PARAMETER SWEEP.  Cycle-detection rate as a function of (m,N),
       distribution of observed periods.

EXPECTED OUTCOMES
   TRUE  : d(prec) → 0 ~ 2^{-prec}; energy identity holds to machine eps;
           every FP64 'no_cycle_found' orbit becomes periodic at prec=400.
   FALSE : at least one orbit's closure distance stays bounded > 0 across
           all precisions, with non-eventually-periodic symbol sequence.
"""")

# ============================================================================
# 2.  CORE DYNAMICS  (FP64 and mpmath)
# ============================================================================
def step_fp(w, M):
    edges = w @ M.astype(np.float64)
    j = int(np.argmax(edges))                    # numpy returns first
        occurrence
    eps = 0.5 * (1.0 - edges[j])
    if not (1e-14 < eps < 1 - 1e-14):
        return None
    alpha = 0.5 * np.log((1 - eps) / eps)
    f = np.exp(-alpha * M[:, j])
    w2 = w * f
    Z = w2.sum()
    return w2 / Z, j, eps, alpha, Z
```

```
def run_fp(M, w0, T):
    w = w0.astype(np.float64).copy()
    m = len(w)
    traj = np.empty((T + 1, m)); traj[0] = w
    syms = np.zeros(T, dtype=int)
    Zs = np.zeros(T); alphas = np.zeros(T)
    for t in range(T):
        out = step_fp(w, M)
        if out is None:
            return traj[: t + 1], syms[:t], Zs[:t], alphas[:t], "degenerate"
        w, j, eps, alpha, Z = out
        traj[t + 1] = w; syms[t] = j; Zs[t] = Z; alphas[t] = alpha
    return traj, syms, Zs, alphas, "ran"

def detect_period_fp(traj, syms, tol=1e-8):
    T = len(traj) - 1
    syms_l = list(syms)
    best_d = float("inf"); best_p = -1
    for p in range(1, T // 2 + 1):
        d = float(np.linalg.norm(traj[T] - traj[T - p]))
        if d < best_d: best_d = d
        if d < tol and syms_l[T - p:T] == syms_l[T - 2 * p:T - p]:
            return p, d
    return -1, best_d

def step_mp(w, M):
    m, N = M.shape
    edges = []
    for j in range(N):
        col = M[:, j]
        s = mp.mpf(0)
        for i in range(m):
            s += w[i] * int(col[i])
        edges.append(s)
    best = 0
    for j in range(1, N):
        if edges[j] > edges[best]:
            best = j
    eps = (mp.mpf(1) - edges[best]) / 2
    if eps <= 0 or eps >= 1:
        return None
    al
# ... [truncated]
```

5 Conclusion

The conjecture remains formally open. Numerical experiments were **inconclusive** — neither strong support nor clear counterexamples were found. Further investigation (both formal and empirical) is warranted.