Process gcPBM: End-to-End Tutorial & Pseudocode

> A step-by-step, implementation-ready guide that mirrors your `process_gcPBM.ipynb` notebook. Use this as teaching notes or a blueprint for re-implementation.

---

0) Purpose & Outputs

Goal. Build clean, balanced, ML-ready datasets from gcPBM experiments for the Myc/Max system.

High-level flow.

1. Load universal PBM 8-mer E-scores, gcPBM probe sequences (36-bp), and normalized binding intensities.
2. Join & annotate probes with central motif and flanks.
3. Apply biological/quality filters (e.g., strong flanks removal, central 8-mer dominance).
4. Transform & label (log intensities → classes).
5. Stratified sampling without motif duplication.
6. Export trainable tables.

Typical outputs.

- `dataset.csv` (balanced subset for experiments)
- `exp_data_all.csv` (full labeled table)

---

1) Inputs & Canonical Columns

Input files.

- `*_8mers_*.txt` → universal 8-mer E-scores; columns: `8mer`, `Escore` (names may vary)
- `gcPBM_probe_sequence.txt` → mapping: `Probe_ID`, `SEQUENCE` (36-bp)
- `gcPBM_*_normalized_intensity.txt` → normalized binding; columns: `Probe_ID`, `Intensity` (or similar)

Core working columns.

- `SEQUENCE` (str, len=36)
- `Probe_ID` (key)
- `Intensity` (float)
- `Log Intensity` (float)
- `8mer_center`, `8mer_flank_left`, `8mer_flank_right` (str)
- k-mers for tie-breaking uniqueness: `6mer`, `8mer`, `10mer`, `12mer`
- labels: `label_str` ∈ {`unbound`, `weak`, `strong`}

---

2) Pseudocode: Load & Harmonize

```
LOAD df_8mer FROM "*_8mers_*.txt"
RENAME df_8mer columns → ["8mer", "Escore"] (if needed)

LOAD df_seq FROM "gcPBM_probe_sequence.txt"
// Expect: [Probe_ID, SEQUENCE]

LOAD df_int FROM "gcPBM_*_normalized_intensity.txt"
// Expect: [Probe_ID, Intensity]

// Merge sequences with intensities on Probe_ID
DF = MERGE(df_seq, df_int, on="Probe_ID", how="inner")
ASSERT len(DF) > 0
```

---

3) Pseudocode: Extract Central Motif & Flanks

```
FUNCTION extract_kmers(seq36):
    // Define center window around position(s) used in notebook
    CENTER_START = 14        // example: choose indices consistent with
notebook
    CENTER_LEN   = 8
    LEFT_START   = CENTER_START - 8
    RIGHT_START  = CENTER_START + CENTER_LEN

    8mer_center = seq36[CENTER_START : CENTER_START + CENTER_LEN]
    8mer_flank_left  = seq36[LEFT_START  : LEFT_START + 8]
    8mer_flank_right = seq36[RIGHT_START : RIGHT_START + 8]

    // For uniqueness tie-breakers
    6mer  = seq36[CENTER_START+1 : CENTER_START+1+6]   // example
    10mer = seq36[CENTER_START-1 : CENTER_START-1+10]  // example
    12mer = seq36[CENTER_START-2 : CENTER_START-2+12]  // example

    RETURN {8mer_center, 8mer_flank_left, 8mer_flank_right, 6mer, 8mer_center
as 8mer, 10mer, 12mer}

DF[kmers] = DF.SEQUENCE.apply(extract_kmers)
```

Note: Use the exact indices your notebook uses. The above indices are placeholders—replace with your real windows.

---

4) Pseudocode: Map E-scores to Center & Flanks

```
// Precompute fast lookup from 8mer → Escore
E = DICT(df_8mer["8mer"] → df_8mer["Escore"])  // default value = NaN if not
found

DF["E_center"]      = DF["8mer_center"].map(E)
DF["E_flank_left"]  = DF["8mer_flank_left"].map(E)
DF["E_flank_right"] = DF["8mer_flank_right"].map(E)
```

---

5) Pseudocode: Biological/Quality Filters

Rationale. Ensure that observed binding is attributable to the central motif, not confounded by strong flanks or nearby higher-scoring alternatives.

```
// (A) Remove probes with strong flanks
THRESH_FLANK = 0.3
KEEP_A = (ABS(DF.E_flank_left)  < THRESH_FLANK) AND (ABS(DF.E_flank_right) <
THRESH_FLANK)
DF = DF[KEEP_A]

// (B) Enforce central-8mer dominance relative to immediate neighbors
// Example approach: compare central E-score against flanking windows or
alternative 8-mers overlapping the center.
FUNCTION neighbors_8mers(seq36, center_start, center_len):
    // Return list of overlapping 8-mers spanning center region (e.g.,
positions center_start-2 → center_start+2)
    NEIGHBORS = []
    FOR s FROM (center_start-2) TO (center_start+2):
        IF s >= 0 AND s+8 <= 36:
            NEIGHBORS.APPEND(seq36[s : s+8])
    RETURN NEIGHBORS

DF["E_neighbors_max"] = DF.SEQUENCE.apply(
    LAMBDA seq36: MAX( E.get(m, -INF) FOR m IN neighbors_8mers(seq36,
CENTER_START, CENTER_LEN) )
)
KEEP_B = (DF.E_center >= DF.E_neighbors_max)
DF = DF[KEEP_B]

ASSERT len(DF) > 0
```

---

6) Pseudocode: Transform & Label

```
// (A) Log-transform for normalization & separation
EPS = 1e-9
DF["Log Intensity"] = LOG(DF["Intensity"] + EPS)

// (B) Choose thresholds (match notebook’s values!)
T_WEAK  = 7.7     // example
T_STRONG = 9.0    // example

FUNCTION relabel(logI):
    IF logI <= T_WEAK:        RETURN "unbound"
    ELSE IF logI < T_STRONG:  RETURN "weak"
    ELSE:                     RETURN "strong"

DF["label_str"] = DF["Log Intensity"].apply(relabel)

// Optional numeric labels
MAP = {"unbound":0, "weak":1, "strong":2}
DF["label_id"] = DF["label_str"].map(MAP)
```

> **Important:** Use exactly the threshold values and logic from your notebook to reproduce figures and counts.

---

7) Pseudocode: Visualization (Optional)

```
PLOT histogram of DF["Log Intensity"], color by label_str
DRAW vertical lines at T_WEAK and T_STRONG
ANNOTATE counts per class
SAVE figure(s)
```

---

8) Pseudocode: Stratified Sampling Without Motif Reuse

Goals. Balanced per-class sample; even coverage across intensity bins; avoid repeated motifs (k-mers) to reduce sequence redundancy.

```
// (A) Make 0.1-wide bins on Log Intensity
BIN_WIDTH = 0.1
DF["log_bin"] = FLOOR( DF["Log Intensity"] / BIN_WIDTH ) * BIN_WIDTH

// (B) Partition by class
D_unbound = DF[ DF.label_str == "unbound" ]
D_weak    = DF[ DF.label_str == "weak" ]
D_strong  = DF[ DF.label_str == "strong" ]
```

```
// (C) Define uniqueness constraints and tie-break order of k-mers
KMER_KEYS = ["8mer", "10mer", "12mer", "6mer"]  // match notebook’s actual
order

FUNCTION sample_stratified_unique(D, n_samples):
    // Initialize
    SELECTED = EMPTY LIST
    SEEN = DICT()  // key → set of observed kmers per KMER_KEYS
    FOR k IN KMER_KEYS: SEEN[k] = EMPTY_SET

    // Group by log bins for even coverage
    GROUPS = GROUPBY(D, by="log_bin")
    ORDERED_BINS = ROUND_ROBIN(GROUPS.keys())

    // Iterate until we hit n_samples or exhaust
    WHILE len(SELECTED) < n_samples AND GROUPS not exhausted:
        FOR b IN ORDERED_BINS:
            CANDIDATES = SHUFFLE(GROUPS[b])
            FOUND = FALSE
            FOR row IN CANDIDATES:
                // Check uniqueness across all k-mer keys
                UNIQUE = TRUE
                FOR k IN KMER_KEYS:
                    IF row[k] IN SEEN[k]:
                        UNIQUE = FALSE; BREAK
                IF NOT UNIQUE: CONTINUE

                // Accept & mark kmers
                SELECTED.APPEND(row)
                FOR k IN KMER_KEYS: SEEN[k].ADD(row[k])
                FOUND = TRUE
                REMOVE row FROM GROUPS[b]
                BREAK
            // If bin exhausted, skip silently
            IF len(SELECTED) == n_samples: BREAK

    RETURN TO_DATAFRAME(SELECTED)

// (D) Execute sampling per class
N_unbound = <set from notebook>
N_weak    = <set from notebook>
N_strong  = <set from notebook>

S_unbound = sample_stratified_unique(D_unbound, N_unbound)
S_weak    = sample_stratified_unique(D_weak,    N_weak)
S_strong  = sample_stratified_unique(D_strong,  N_strong)

S = CONCAT_ROWS([S_unbound, S_weak, S_strong])
ASSERT len(S) == N_unbound + N_weak + N_strong
```

Notes.

- If bins run dry, the loop naturally shifts to other bins.
- The *tie-break order* among k-mers should mirror the notebook.
- For reproducibility, set a fixed RNG seed before shuffling.

---

9) Pseudocode: Train/Test Split (Optional)

```
SET random seed
S_train, S_test = STRATIFIED_SPLIT(S, by="label_str", test_size=0.2)
```

---

10) Pseudocode: Final Assembly & Export

```
// (A) Minimal ML table
EXP = DF[["SEQUENCE", "Log Intensity", "label_str", "label_id"]]
RENAME EXP columns: {
    "SEQUENCE"      → "sequence",
    "Log Intensity" → "bind_avg"
}
SAVE EXP TO "exp_data_all.csv"

// (B) Balanced subset for experiments
SAVE S TO "dataset.csv"

// (C) (Optional) Save sampling metadata
SAVE class counts, bin counts, and RNG seed to a JSON/YAML sidecar
```

---

11) Sanity Checks (Recommended)

```
ASSERT no NA in required columns
ASSERT all sequences length == 36
ASSERT min/max/mean(Log Intensity) within expected ranges
ASSERT class balance in S matches targets
ASSERT uniqueness: no duplicated k-mers across S per KMER_KEYS
```

---

12) Reproducibility Tips

- Fix **random seeds** for all sampling steps.
- Log configuration → thresholds, window indices, bin width, sample counts.
- Version control the input files or their checksums.

13) Common Variations

- Adjust `T_WEAK` & `T_STRONG` depending on your histograms.
- Swap k-mer priority order if you need different motif diversity.
- Increase `BIN_WIDTH` if data are sparse.

14) Quick Checklist

-

```
```