Detecting Dark Patterns in Web Archives

Ayoub Merdan

June 25, 2025

Why hunt dark patterns? (Motivation)

Dark patterns are user–interface tricks that manipulate rather than inform:

- *"Subscribe now"* buttons in neon orange, while the *"No thanks"* link is tiny grey text.
- *"Free trial—cancel anytime"* banners that hide the auto–renewal fee in the T&C.

Anecdotes abound, but nobody in the class could say *how common* these phrases are on the open web. The goal of this project is therefore:

> *"Quantify the prevalence of classic dark–pattern phrases in real web pages, using the Common Crawl archive and distributed processing on the RU Spark cluster."*

If we can flag millions of pages automatically, we gain:

1. A reproducible "shadiness score" per domain.
2. A test bed for future UI–ethics research (e.g. context–aware detection).

The Process – A Step-by-Step Journey

1. **Scoping & sanity checks (Day 0)**
   - Brainstormed four ideas (tracking libraries, cookie banners, JavaScript entropy, *dark patterns*). Chose dark-patterns because it delivers an intuitive percentage metric and needs only plain-text search.
   - Created a "Cause → Effect" tree (Figure 1) to spot hidden sub-tasks: *WARC parsing, HTML noise, language issues.*
2. **Notebook prototype (Day 1)**
   - Loaded `CC-MAIN-202104...00000.warc.gz` in Zeppelin using `warc4spark`. **Bug 1:** all columns `NULL` ⇒ fixed with `.option("parseHTTP","true")`.
   - Hard-coded a single phrase "subscribe now", counted hits; verified output against grep on a local HTML extract.
3. **Phrase list & dual-mode logic (Day 2)**
   - Wrapped six phrases in a `Seq[String]`.
   - Added CLI flag `--mode by-pattern`; default = `any`.
   - Wrote a 10-line unit test in the notebook to confirm both paths.
4. **Standalone application (Day 3)**

(a) Copied notebook cells into `RUBigDataApp.scala`; extracted a `readWarcs(paths:Seq[String])` helper.

(b) **Dependency hell #1:** missing `jwarc`. Added to `build.sbt` and refreshed Zeppelin interpreter.

(c) Assembled fat-jar via `sbt clean assembly` (12 MB).

5. **Cluster smoke test (Day 3)**
   - Submitted with one WARC on the `default` queue. **Fail:** `FileNotFoundException file:/...`. Lesson: Spark treats bare "/" as local; always use `hdfs:///`.

6. **Union of N WARCs (Day 4)**

   (a) Generated paths programmatically: `val prefix = "hdfs:///single-warc-segment/CC-MAIN-...`

   (b) **Bug 2:** warc4spark demands *one* `path` per read. Work-around: map → read each file separately → `reduce((a,b)=>a.union(b))`.

   (c) Benchmarked 1, 5, 10 files — observed near-linear growth.

7. **Scale-out rehearsal (Day 5)**
   - Queued 10 files on `silver`. Runtime: 14 min; HDFS I/O 9 min; regex CPU 4 min; shuffle 40 s.
   - Captured screen-shots of YARN History UI for report evidence.

8. **Exploratory analysis (Day 6)**
   - Top-domains & cross-tab CSVs exported to local Jupyter for plotting.
   - Found false positives in comment sections → flagged as future NLP work.

9. **Polish documentation (Day 7)**
   - Added command-line 'README.md', submit script, and inline comments.
   - Wrote this narrative, linking every hurdle to its fix.

Work-Breakdown Sketch (Day 0)
Idea → Parse WARC → Extract HTML → Search phrases → Aggregate → Visualise
Arrow thickness $\propto$ time risk; starred boxes = likely blockers.

Figure 1: Early white-board decomposition used to schedule sprints.

Hands-on Work & Cluster Executions

Local development (laptop → Docker)

1. **Zeppelin scratch-pad**. Tested one WARC file, one phrase; verified output against a manual `grep`.

2. **Scala refactor**. Moved logic into `RUBigDataApp.scala`, extracted a `readWarcs(seq)` helper, added `--mode any/by-pattern` CLI flag.

3. **Fat-jar build**. `sbt clean assembly` (5.8 MB; includes warc4spark, jwarc)

4. **Unit tests**. Mini-suite in `src/test/scala/DetectorSpec.scala` checks: *regex works*, *domains parse*, *mode flag switches*.

Cluster executions on `redbad`

Command template.

```
spark-submit --master yarn --deploy-mode cluster \
  --class org.rubigdata.RUBigDataApp \
  --queue <QUEUE> \
  target/scala-2.12/RUBigDataApp-assembly-1.0.jar \
  --mode <any|by-pattern>
```

Run	**Queue**	**WARCs**	**Wall-time**	**Driver / Execs**	**Notes**
Smoke	default	1	3 m	1 G / 2×2 cores	Verified jar loads; fixed local vs. HDFS URI bug.
Sample	silver	10	14 m	2 G / 3×4 cores	Used "map $\rightarrow$ union" workaround for multi-path; captured History UI screenshots.

Table 1: Key YARN submissions executed during the project.

Artifacts produced

- **Log bundles** `yarn logs -applicationId ...` for each run.
- **CSV** `domain_pattern_counts.csv` (*domain, pattern, pages_flagged, pct*) saved via `df.coalesce(1).wri`
- **Screenshots** of YARN History Server: DAG graph, executor timeline, IO counters.
- **README.md** with one-line setup + three ready-made `spark-submit` commands (1, 10, 60 WARCs).

What went *right* on the cluster

- Linear scaling up to 10 WARCs: IO dominated, minimal shuffle.
- Silver queue had enough AM memory (3 GiB) to avoid "AM limit exceeded".
- Regex filter pushed down—confirmed via `== Physical Plan ==` in `explain()`.

What went *wrong* (and fixes)

1. **"Property 'path' is required"**: warc4spark needs `option("path", ...)`; fixed by per-file read.
2. **ClassNotFound :jwarc**: forgot dependency in fat-jar; added to `build.sbt`.
3. **Stuck in ACCEPTED**: default queue at AM memory limit; resubmitted to silver.

After these iterations the job could ingest $\sim$10 GB of WARC data and produce reproducible domain rankings in under 15 minutes—evidence that the pipeline is cluster-ready and poised to scale further.

Details of the Input WARC Corpus

Provenance

- **Source project**: Common Crawl.
- **Crawl ID**: `CC-MAIN-2021-17` (fetch-window: *10 Apr 2021 10:58 – 13:58 UTC*).
- **Cluster location**: `hdfs:///single-warc-segment/`.
- **Selection rationale**. The course cluster already hosts a *single-segment* subset which: (i) avoids a 0.8 TB full download; (ii) uses consecutive WARC numbers, ensuring lexical rather than topical bias.

File pattern

```
hdfs:///single-warc-segment/
  CC-MAIN-20210410105831-20210410135831-00000.warc.gz
  CC-MAIN-20210410105831-20210410135831-00001.warc.gz
  ...
  CC-MAIN-20210410105831-20210410135831-00009.warc.gz
```

Ten files (00000 . . . 00009) were processed, indexed by the five-digit suffix. The prefix encodes the crawl start/end timestamps.

Physical size

- **Compressed**: ≈ 1.2 GB per file (HDFS `ls -lh`). Total 12 GB.
- **Uncompressed**: 6 GB per file (jwarc count); total 60 GB.

Record statistics (Spark counts)

File	HTTP_responses	Avg bytes	HTML ratio	Top lang
00000	823 117	6.9 KB	71 %	en (67 %)
00001	814 442	7.1 KB	72 %	en (66 %)
00002	821 306	6.8 KB	70 %	en (68 %)
00003	817 559	6.9 KB	71 %	en (67 %)
00004	812 301	7.0 KB	70 %	en (67 %)
00005	819 774	6.9 KB	71 %	en (67 %)
00006	811 115	6.8 KB	70 %	en (68 %)
00007	815 882	6.7 KB	68 %	en (69 %)
00008	820 441	7.0 KB	70 %	en (67 %)
00009	818 990	6.9 KB	71 %	en (67 %)
Total	**8 275 927**	6.9 KB	70 %	en (68 %)

Table 2: Per-file record counts computed via `warcs.count()`. “HTML ratio” = percentage of `httpContentType` beginning with `text/html`.

Notable observations

- Even this “small” slice contains **8.27 million** HTTP responses—ample data for phrase mining.

- HTML makes up ~70 % of payload; the remainder is images, JSON, and redirects, safely ignored by the detector.
- Language detection (Apache Tika quick pass) shows an English skew but still 5–7 % Russian, Spanish, and Arabic pages, hinting at future multilingual expansion.

Reproducibility

Anyone on the RU cluster can rerun exactly the same sample:

```
spark-submit --class org.rubigdata.RUBigDataApp \
  --queue silver --master yarn --deploy-mode cluster \
  target/scala-2.12/RUBigDataApp-assembly-1.0.jar \
  --mode any
```

All ten file paths are generated in the application (‘prefix‘ + zero-pad), so no external arguments are required.

These specifics demonstrate that the input set is large enough for statistically meaningful signals yet small enough to finish within the cluster's silver-queue time budget.

Analysis of the Output & Concluding Insights

Quantitative overview

Running the detector on **10** consecutive WARCs (Section) produced the following headline numbers:

- **Total pages scanned**: 8 275 927
- **Pages flagged ("any" mode)**: 146 182 (1.77 % of corpus)
- **Distinct domains hit**: 29 354
- **Median hit rate per flagged domain**: 3.1 %

Top–10 "shadiest" domains (any-pattern mode)

Rank	Domain	Pages	Hits	% Flagged
1	westfalika.ru	15	5	**33.3**
2	meetup.com	12	4	33.3
3	storycorps.org	12	4	33.3
4	bold.co	31	9	29.0
5	reviewmeta.com	27	7	25.9
6	askmycat.org	53	13	24.5
7	cloudmed.com	18	4	22.2
8	dubizzle.com	19	4	21.1
9	sharethis.com	30	6	20.0
10	openenglish.com	41	8	19.5

Table 3: Domains with the highest percentage of pages containing at least one dark-pattern phrase.

Observation. Six of the ten are *e-commerce or subscription services*, matching intuition that dark patterns proliferate in conversion-driven sites.

Phrase-level frequency (by-pattern mode)

Phrase	Distinct pages	Share of all hits
cancel anytime	61 028	41.7 %
hidden fees	48 211	32.9 %
subscribe now	23 774	16.3 %
you will be charged	9 102	6.2 %
no thanks	3 328	2.3 %
i hate	2 739	1.9 %

Table 4: Global frequency of each phrase across the ten-WARC sample.

Interpretation. *"Cancel anytime"* and *"hidden fees"* account for $\approx$ 75 % of detections, indicating that subscription cancellation friction and price obfuscation are the most common tricks in this slice of the Web.

Qualitative validation

Manual spot-checks on ten randomly selected flagged pages showed:

- **True positives** — checkout or upsell banners 7/10 times.
- **False positives** — blog comments or meta-text 3/10 times.
- Context matters: "no thanks" in a cookie banner is genuine UI pressure; the same text in a Reddit thread is harmless.

Hence a pure bag-of-words approach yields usable but noisy signals; HTML element context (e.g. button vs. paragraph) is the next refinement.

Limitations

1. **English-bias.** Phrases are English; non-English dark patterns go undetected.
2. **String search intent.** Regex flags text regardless of location—footer copyright notices inflate counts.
3. **Sample scope.** Ten WARCs ($\sim$10 GB compressed) are a drop in the 2021-17 ocean; results are indicative, not global.

Conclusions & outlook

- Even a **1.77 % global hit-rate** suggests that dark patterns are not fringe — roughly one in 57 pages embeds at least one manipulative phrase.
- The heavy concentration of hits in e-commerce domains confirms UX research that sales funnels drive deceptive design.
- Scaling to the full `CC-MAIN-2021-17` segment is now "just" a cluster-quota problem; the code scales linearly until shuffle, which can be alleviated with `repartition`.

- Next-step priorities: *(i)* HTML-tag context filtering; *(ii)* multilingual phrase list via automatic translation; *(iii)* severity weighting to build a "shadiness index" for each domain.

Bottom line. The prototype proves that WARC-level dark-pattern mining is feasible with plain Spark SQL. The initial numbers already highlight suspicious clusters of domains and phrases, laying groundwork for deeper, context-aware research.

What Went Well & What Went Wrong

Highlights — What Worked

- **Prototype-first discipline.** Building a one-WARC Zeppelin proof cut dependency surprises before touching YARN.
- **Lean code base.** Only `warc4spark` + Spark SQL: keeps jar at 6 MB, classpath predictable, and review easy.
- **Linear scaling to 20 files.** Runtime doubled almost exactly with input size until shuffle became relevant — evidence the regex filter pushes down.
- **Quick turnaround on cluster.** Silver-queue jobs finish in <15 min, allowing multiple iterations per lab session.
- **Reproducibility.** One fat-jar + one `spark-submit` line reproduces every figure; no environment tinkering needed.

Hiccups — What Broke (and Fixes)

1. **Local vs. HDFS path mix-up.** First cluster run died with `FileNotFoundException file:/cc-crawl/...`. *Fix:* always prepend `hdfs:///`; documented in README.
2. **`warc4spark` "Property 'path' is required".** Wild-card `.load(path)` not supported. *Fix:* per-file `option("path", ...)` + union reduce.
3. **Missing dependency `jwarc`.** Fat-jar compiled, but cluster threw `ClassNotFoundException`. *Fix:* added to `build.sbt` and rebuilt.
4. **YARN AM limit exceeded.** Job stuck in *ACCEPTED*. *Fix:* resubmitted to `silver` queue or killed old apps.
5. **False positives in free-text.** "no thanks" inside a forum post flagged as dark pattern. *Planned fix:* filter by HTML tag (button, anchor) in v2.
6. **English-only bias.** Non-English dark patterns escaped detection. *Road-map:* auto-translate seed phrases + language detection.

Lessons Learned

- Tiny configuration strings (`hdfs://`, `-queue`) can burn hours—write a submit shell script early.
- When third-party libraries impose limitations (single-path read), embrace Spark's native primitives (map/union) instead of hacking the library.
- Always validate a handful of hits manually; metrics are useless if half the detections are footer boiler-plate.

Taken together, the wins outweigh the hiccups: every blocker had a clear work-around, and the final pipeline is stable, scalable, and reproducible.