Mini AI Pipeline Project — Phishing URL Detection

Changhyun Lee (2024149026)

1 Introduction

I built a small end-to-end pipeline to classify a URL string as either benign (0) or phishing (1). I chose this task because it is realistic (phishing links are a frequent attack vector) and the input format is simple enough to iterate quickly with clear evaluation. To make the comparison between methods straightforward, I used a balanced subset from a public dataset (1,046 benign / 1,046 phishing; total 2,092).

The project compares three approaches: (i) a rule-based heuristic baseline with score thresholding, (ii) a lightweight learned baseline using handcrafted URL statistics with Logistic Regression, and (iii) an embedding-based pipeline using SentenceTransformer (MiniLM) embeddings with Logistic Regression. During development, I observed that allowing threshold = 0 in the heuristic makes the model predict nearly everything as phishing on my validation split, so I restricted threshold candidates to start from 1 to avoid this degenerate behavior.

2 Task Definition

- **Task description:** Binary classification of URLs into {benign (0), phishing (1)}.
- **Motivation:** A practical detector should catch phishing links (recall) while limiting false alarms (precision).
- **Input / Output:** Input is a URL string (e.g., `dehomefurniture.com/files/get_msg.html`); output is a label in {0,1}.
- **Success criteria:** Evaluate on a held-out test set using accuracy and phishing-class (label 1) precision/recall/F1. I emphasize phishing-class F1 and recall because missing phishing cases is costly.

3 Methods

3.1 Naïve Baseline

Baseline 1: Heuristic scoring + threshold tuning

- **Method description:** I assign an integer score to each URL using simple, interpretable signals: long length (>50), presence of special characters (e.g., `@`), many hyphens/dots, high digit ratio, punycode (`xn--`), and suspicious keyword tokens (e.g., `login`, `verify`, `bank`). A URL is predicted as phishing if `score` $\geq$ `threshold`. The threshold is tuned on a validation split within the training set by maximizing phishing-class F1 (tie-break by phishing recall). In practice, I excluded threshold = 0 because it produced a degenerate predictor (almost all outputs become 1 on validation).

- **Why naïve:** The scoring function is manually designed and limited to a small set of surface patterns; it does not learn which patterns matter from data.
- **Likely failure modes:** (1) Benign URLs with long paths or many subdomains can look "suspicious" under simple counting rules (false positives). (2) Short phishing URLs that avoid obvious tokens/symbols can slip through (false negatives). (3) The keyword list is narrow and attackers can avoid those exact strings.

Baseline 2: Handcrafted URL features + Logistic Regression

- **Method description:** I featurized each URL into numeric counts (length, counts of '.', '-', '@', '?', '=', '%', `https://` prefix, presence of `xn--`, number of digits), then trained Logistic Regression on these features.
- **Why naïve:** Even though it is trained, the representation is still limited to a small set of URL statistics and cannot capture richer string patterns.
- **Likely failure modes:** Phishing URLs that mimic benign statistics (similar lengths and symbol counts) may evade detection, and benign but unusual URLs may be flagged.

3.2 AI Pipeline

SentenceTransformer embeddings (MiniLM) + Logistic Regression

- **Models used:** Sentence embedding model `sentence-transformers/all-MiniLM-L6-v2` and Logistic Regression.
- **Pipeline stages:**
  1. **Preprocessing:** Use the raw URL string as input (only basic string cleanup done in data loading).
  2. **Embedding:** Convert each URL into a dense vector using the pre-trained Sentence-Transformer encoder.
  3. **Decision:** Train Logistic Regression on embeddings and predict labels on the test set.
  4. **Evaluation:** Compare to baselines on the same test split using the same classification metrics.
- **Design choices and justification:** I chose this pipeline to test whether a generic text embedding model can capture useful regularities in URL strings without manual feature engineering. Logistic Regression keeps training lightweight, so the performance difference mainly reflects the representation quality (handcrafted counts vs. learned embeddings).

4 Experiments

4.1 Datasets

Dataset Description

- **Source:** Dataset `shawhin/phishing-site-classification` from the `datasets` library.
- **Total examples used:** 2,092 URLs after preprocessing and class balancing (1,046 per class).
- **Splits:** Stratified 80/20 train/test split: 1,673 train and 419 test. The test set contains 210 benign and 209 phishing. For Baseline 1 threshold tuning, the training set is further split into train_sub/validation (75/25), stratified.

- **Preprocessing steps:** Normalize column names, detect URL/label columns, drop missing rows, strip whitespace, remove duplicate URLs, then downsample each class to the same size and shuffle with a fixed random seed.

4.2 Metrics

I report accuracy and per-class precision/recall/F1. I focus on phishing-class (label 1) F1 and recall because they measure the trade-off between catching phishing links and avoiding missed detections.

4.3 Results

Quantitative results

All methods are evaluated on the same held-out test set.

Method	Accuracy	F1 (Phishing=1)
Baseline 1: Heuristic (threshold tuned)	0.5704	0.5833
Baseline 2: Handcrafted features + LR	0.5919	0.5535
AI Pipeline: MiniLM embeddings + LR	0.8329	0.8325

For reference, the confusion matrices (rows=true, cols=pred) were:

- Baseline 1: $\begin{bmatrix} 113 & 97 \\ 83 & 126 \end{bmatrix}$
- Baseline 2: $\begin{bmatrix} 142 & 68 \\ 103 & 106 \end{bmatrix}$
- AI Pipeline: $\begin{bmatrix} 175 & 35 \\ 35 & 174 \end{bmatrix}$

Qualitative examples

Below are five representative test examples (URL, true label y, Baseline1 prediction $b1$, AI prediction ai). When explaining why Baseline 1 fired, I am making an *inductive* inference based on the heuristic rules (length/dots/digits/keywords), not from a per-URL feature dump.

- `www.cs.waikato.ac.nz/~ihw/mg.html` ($y = 0$, $b1 = 1$, $ai = 0$): Baseline 1 likely overreacts to structure (multiple dots and a path), while the embedding model predicts benign.
- `torrentreactor.net/torrents/...` ($y = 0$, $b1 = 1$, $ai = 0$): A long path and symbols can increase the heuristic score even for benign content pages.
- `dehomefurniture.com/files/get_msg.html` ($y = 1$, $b1 = 0$, $ai = 1$): A phishing URL can look “clean” (few obvious tokens), so the heuristic misses it, while the embedding-based model catches it.
- `windowsandpaperwalls.wordpress.com/` ($y = 0$, $b1 = 0$, $ai = 1$): The AI pipeline produces a false positive; one plausible reason is that some benign hosting domains share surface patterns with phishing in the dataset.
- `catlong.com/jhg45s]` ($y = 1$, $b1 = 0$, $ai = 0$): Both methods miss this case; it suggests that short, low-signal URLs remain challenging for both rule-based signals and embeddings.

5 Reflection and Limitations

The MiniLM-embedding pipeline clearly outperformed the baselines (accuracy 0.8329, phishing F1 0.8325), which suggests representation quality mattered more than classifier complexity. Baseline 1 was transparent, but I had to handle a degenerate case during tuning (threshold = 0 making almost-all-phishing predictions), and its rules were not robust. Baseline 2 slightly improved accuracy over Baseline 1, but phishing F1 decreased, showing that simple URL statistics can shift the error trade-off in an undesirable direction. A limitation is that I downsampled to a perfectly balanced dataset, which makes comparison easy but may not match real-world class ratios. Threshold tuning used a single validation split; cross-validation would likely produce a more stable threshold choice. Even the AI pipeline still makes both false positives and false negatives, so it is not reliable enough for deployment without further testing. With more time, I would evaluate under imbalanced settings and try URL-specialized or character-level models to better capture string-level patterns. For reproducibility, I fixed random seeds and used stratified splits so that results are stable under the same environment.