Phishing Detector: Data Preprocessing & Consolidation Guide

This document outlines the step-by-step procedure for cleaning, consolidating, and preprocessing multiple email datasets. It presents two distinct pipelines: **Pipeline A** for traditional "bag-of-words" models (e.g., SVM, Random Forest) and **Pipeline B** for context-aware transformer models (e.g., BERT).

Step 1: Data Consolidation & Label Standardization (Common to Both Pipelines)

- **Objective:** To combine all disparate data sources into a single, uniform, and auditable dataset with consistent labels.
- **Load All Datasets:** Ingest all source files (CSVs, text files, etc.) into a single master data frame.
- **Source & Provenance Tracking:** To maintain reproducibility, add metadata columns during ingestion.
  - source_name: The dataset or corpus of origin (e.g., "PhishCorpus_2023").
  - original_file: The specific filename the record came from.
  - record_id: A unique ID, such as the original index.
- **Standardize Labels:** Map all variations of class labels to a single, consistent binary format.
  - Phishing Email, 1, spam $\rightarrow$ **1** (Phishing)
  - Safe Email, 0, ham $\rightarrow$ **0** (Legitimate)
- **Label Verification (Optional):** Perform sanity checks to flag potentially mislabeled data for review. This can include keyword-based heuristics or using a simple pre-trained model to find high-confidence disagreements.
- **Consolidate Structure:** Ensure the final master dataset has a consistent structure, primarily text (containing the full email content) and label (containing the 0 or 1 classification), along with the new provenance columns.
- **Output:** A single master CSV file (e.g., master_dataset_raw.csv).

At this point, we must select a modeling path. The subsequent cleaning and preprocessing steps are mutually exclusive.

PIPELINE A: For Traditional ML Models (e.g., Random Forest, SVM)

This pipeline performs aggressive cleaning to create a "bag-of-words" representation.

Step A1: Text Cleaning & Normalization

- **Objective:** To remove non-textual "noise" and artifacts from the raw email content.
- **Fix Encoding Artifacts:** The raw data contains "quoted-printable" encoding remnants. These must be fixed to reconstruct the original text.

    - **Line Breaks:** Replace soft line breaks (e.g., =\n or =\s) with an empty string ("") to join words.
    - **Character Codes:** Replace or decode character codes (e.g., ==2E is a period .).
- **Strip HTML Tags:** Use a library like BeautifulSoup to parse the text and remove all HTML tags (e.g., <p>, <b>,), retaining only the human-readable text.
- **Replace URLs and Email Addresses:** Replace them with special, uniform tokens.
    - Replace all URLs (starting with http://, https://, or www.) with a special token: [URL].
    - Replace all email addresses with a special token: [EMAIL].
- **Normalize Text (Aggressive):**
    - Convert all text to **lowercase**.
    - Expand common contractions (e.g., "can't" $\rightarrow$ "cannot").
    - Replace digits with a generic token (e.g., \d+ $\rightarrow$ [NUM]) to retain their semantic presence without increasing feature sparsity.
    - Normalize all whitespace by replacing multiple spaces, newlines (\n), and tabs (\t) with a **single space**.
    - Remove any other **non-ASCII characters or punctuation** that remain.
- **Text Integrity Validation:** After cleaning, verify that the text column is not empty or filled only with placeholders (e.g., df = df[df['text'].str.strip().str.len() > 0]).

Step A2: Deduplication

- **Objective:** To prevent data leakage and model bias by ensuring no duplicate records exist.
- **Identify Duplicates:** Scan the **cleaned text column** from Step A1 for identical entries.
- **Remove Duplicates:** Delete all but one instance of any exact-match duplicate. This is critical for ensuring your model is not tested on data it has already seen during training.

Step A3: Outlier & Length Filtering

- **Objective:** To remove abnormally short or long documents that can skew the model.
- **Filter by Length:** Calculate the token count for each document.
- **Remove Outliers:** Filter out samples with fewer than N tokens (e.g., < 5) or more than M tokens (e.g., > 2000). These thresholds should be set after analyzing the distribution of text lengths.

Step A4: Data Splitting (Train-Test Split)

- **Objective:** To create separate, stratified datasets for training and final, unbiased validation.
- **Action:** Perform a single **80/20 stratified random split** on the cleaned, deduplicated, and filtered dataset from Step A3.
    - **80% Development Set (X_train, y_train):** This partition is used for *all* model training, cross-validation, and hyperparameter optimization.
    - **20% Held-Out Test Set (X_test, y_test):** This partition is locked away and is **not**

touched until the very end for a single, final evaluation.

- **CRITICAL:** All subsequent steps (Vectorization, Imbalance Handling) are fit *only* on the **80% Development Set**. The 20% Test Set is only *transformed* using the artifacts (e.g., the fitted vectorizer) from the development set.

Step A5: Vectorization & Imbalance Handling (via CV Pipeline)

- **Objective:** To convert text to TF-IDF vectors, handle class imbalance, and find the best model, all within a cross-validation loop to prevent data leakage.
- **Methodology:** Use a scikit-learn Pipeline combined with GridSearchCV. This is the most robust method.
- **Pipeline Steps:**
  1. **Vectorization (TF-IDF):** This is your feature engineering. The TfidfVectorizer is the first step. Here you will:
     - Remove **Stop Words** (e.g., stop_words='english').
     - Apply **Lemmatization** (e.g., TfidfVectorizer(tokenizer=MyLemmaTokenizer)).
     - **Curate Vocabulary:** Use TfidfVectorizer parameters like min_df=5 (remove terms appearing in < 5 documents) and max_df=0.9 (remove terms appearing in > 90% of documents) to reduce noise and dimensionality.
  2. **Imbalance Handling (SMOTE):** Use imbalanced-learn's SMOTE as the second step.
  3. **Classifier:** The model (e.g., RandomForestClassifier, SVC) is the final step.
- **Grid Search:** GridSearchCV is then wrapped around this *entire* pipeline and **fit on the 80% Development Set**. It will automatically apply Stratified K-Fold cross-validation.
- **Justification:** This process correctly applies SMOTE *only* to the "mini-train" data within each fold, preventing synthetic data from leaking into the "mini-validation" fold. This is the correct, leak-free way to tune and handle imbalance simultaneously.

PIPELINE B: For Transformer Models (e.g., BERT, RoBERTa)

This pipeline performs a gentle clean to preserve the context (case, punctuation) that transformers rely on.

Step B1: Text Cleaning (Transformer Path)

- **Objective:** To remove non-textual noise *without* destroying the contextual information.
- **Fix Encoding Artifacts:** (Same as A1) Decode "quoted-printable" text.
- **Strip HTML Tags:** (Same as A1) Use BeautifulSoup to remove HTML.
- **Replace URLs and Email Addresses:** (Same as A1) Replace with [URL] and [EMAIL] tokens.
- **Normalize Whitespace:** Replace \n, \t and other whitespace with a **single space**.
- **CRITICAL: Do NOT** convert to lowercase. **Do NOT** remove punctuation.
- **Text Integrity Validation:** (Same as A1) After cleaning, verify that the text column is not empty or filled only with placeholders.

Step B2: Deduplication

- **Objective:** (Same as A2) To prevent data leakage.
- **Identify Duplicates:** Scan the **transformer-cleaned text column** from Step B1 for identical entries.
- **Remove Duplicates:** Delete all but one instance of any exact-match duplicate.

Step B3: Outlier & Length Filtering

- **Objective:** To remove documents that do not fit the model's constraints.
- **Filter by Length:** Transformer models have a hard token limit (e.g., 512 for BERT). Documents exceeding this must be truncated or split.
- **Remove Outliers:** Filter out samples with fewer than N tokens (e.g., < 5) to avoid processing empty or near-empty inputs.

Step B4: Data Splitting (Train-Test Split)

- **Objective:** (Same as A4) To create stratified, independent datasets.
- **Action:** Perform a single **80/20 stratified random split** on the cleaned, deduplicated, and filtered dataset from Step B3.
  - **80% Development Set (X_train, y_train):** Used for all training and tuning.
  - **20% Held-Out Test Set (X_test, y_test):** Locked away for final evaluation.
- **CRITICAL:** (Same as A4) All fitting is done on the 80% set.

Step B5: Tokenization & Formatting

- **Objective:** To convert the semi-clean text into the specific input format for the transformer.
- **Tokenization:** Use the **specific, pre-trained tokenizer** associated with your chosen model (e.g., BertTokenizer).
- **Vectorization:** The tokenizer will handle the conversion of text into the required model inputs (e.g., input_ids, attention_mask).
- **Max Length Strategy:** For documents longer than the model's max sequence length (e.g., 512 tokens), apply an intelligent truncation strategy, such as keeping the first 128 tokens and the last 382 tokens (since phishing cues are often at the start or in the signature).
- **Efficiency:** For large datasets, use the tokenizer's batch encoding (batch_encode_plus) or a library like datasets.map() to apply tokenization in parallel.

Step B6: Handling Class Imbalance (Training Set ONLY)

- **Objective:** To prevent the model from ignoring the minority class during training.
- **Methodology:** SMOTE is not ideal here as it works on vectors, not token IDs. The best practice is to handle imbalance in the DataLoader at batch-creation time.
- **Primary Recommendation (WeightedSampler):** Use a WeightedRandomSampler (e.g., in PyTorch). This component is given weights based on the class imbalance. When

building each training batch, it will oversample from the minority (phishing) class, ensuring the model sees a more balanced distribution of data over time.

- **Alternative (Class-Weighted Loss):** A powerful alternative (or supplement) is to use a class-weighted loss function, (e.g., nn.CrossEntropyLoss(weight=...)). This penalizes misclassifications of the minority class more heavily.
- **Optional (Text Augmentation):** On the 80% Development set, light augmentation (e.g., Easy Data Augmentation, back-translation) can be applied *only* to the minority class to create more training examples.

Post-Preprocessing & Model Training Considerations

- **Mixed Precision:** (Pipeline B) When fine-tuning transformers, use mixed-precision training (torch.cuda.amp.autocast()) to speed up training and reduce memory usage.
- **Post-Training Calibration:** For production deployment, model *probabilities* must be reliable. After training, use Platt Scaling or Isotonic Regression on the validation set's outputs to calibrate probabilities.
- **Explainability:** For auditing and analysis, integrate explainability tools.
  - **Pipeline A:** Use **SHAP** or **LIME** on the TF-IDF features.
  - **Pipeline B:** Use **Integrated Gradients** (e.g., via the transformers-interpret or captum libraries).

Final Deliverables & Artifacts

- **Objective:** To ensure the entire experiment is reproducible, auditable, and deployable.
- **Logging:** Use Python's logging module to log dataset sizes, rows dropped at each step, split statistics, and final class balances.
- **Artifact Checklist:** Each pipeline run should generate a set of version-tagged artifacts:
  - **Data:**
    - cleaned_train.csv / cleaned_test.csv: The final, split datasets.
  - **Preprocessing Objects:**
    - (Pipeline A) tfidf_vectorizer.joblib: The fitted TfidfVectorizer.
    - (Pipeline B) tokenizer_config.json: The tokenizer configuration.
  - **Model:**
    - model.pkl or pytorch_model.bin: The final, trained model.
  - **Results & Logs:**
    - training_log.txt: Log of metrics, parameters, and run version.
    - metrics.json: Final evaluation results (precision, recall, F1, ROC-AUC) from the held-out test set.