Aprendizagem Aplicada à Segurança

Text Mining for Spam Detection

Mário Antunes

September 26, 2025

Universidade de Aveiro

Table of Contents i

The Problem: Spam Detection

At its core, **spam detection** is a **binary classification** problem.

The goal is to build a model that reads a piece of text (like an email or SMS) and assigns it one of two labels:

- **Ham:** A legitimate, non-spam message.
- **Spam:** An unwanted, unsolicited message.

To do this, we use **Text Mining** and **Natural Language Processing (NLP)** to teach a machine how to understand and differentiate between these two classes of text.

The Text Mining Pipeline

You cannot just feed raw text to a machine learning model. Models need **numbers**, not words.

The entire process involves a standard “pipeline” to convert messy text into a structured, numerical format that a model can learn from.

1. **Text Pre-processing:** Cleaning and normalizing the raw text.
2. **Feature Extraction (Vectorization):** Converting clean text into numerical features.
3. **Model Training:** Teaching a classification model to see patterns in the numbers.
4. **Model Evaluation:** Checking how well the model performs on unseen data.

Step 1: Text Pre-processing (Cleaning)

The first step is to clean the raw text to remove “noise.”

- **Punctuation Removal:** Characters like `!`, `?`, `.`, and `,` generally don’t add much meaning for this task and are removed. In spam, `!` might be a feature, but it’s often simpler to remove it.
- **Tokenization:** The process of splitting a string of text into a list of individual words, or “tokens.”
  - `"Call you now!"` -> `['Call', 'you', 'now']`
- **Stopword Removal:** Removing highly common words that provide little semantic value for classification (e.g., ‘the’, ‘is’, ‘a’, ‘in’, ‘and’). This makes the dataset smaller and focuses the model on important words.

Step 1 (Cont.): Text Normalization

After cleaning, we normalize words so that different forms of the same word are treated as one.

- **Goal:** Group words like `running`, `ran`, and `runs` into a single root concept: `run`.

This is done in one of two ways:

1. **Stemming**
2. **Lemmatization**

Stemming vs. Lemmatization

Stemming

- A crude, fast, and simple rule-based process.
- It **chops off the end** of words to find a common “stem.”
- **Example:** `running` -> `runn`, `studies` -> `studi`, `argue` -> `argu`
- **Pro:** Very fast.
- **Con:** The resulting “stem” may not be a real word.

Lemmatization

- A more sophisticated, dictionary-based (linguistic) process.
- It finds the actual dictionary root of a word (the “lemma”).
- **Example:** `running` -> `run`, `studies` -> `study`, `was` -> `be`
- **Pro:** More accurate and the results are real words.
- **Con:** Much slower, as it needs to look up words in a dictionary (like WordNet).

Step 2: Feature Extraction (Vectorization)

Now that we have a clean list of tokens for each message, we must convert them into a numerical matrix.

This is the most critical step. The two main methods are:

1. **Bag of Words (BoW)**
2. **TF-IDF (Term Frequency-Inverse Document Frequency)**

Vectorization Method 1: Bag of Words (BoW)

This is the simplest method. It creates a matrix based on word counts.

1. **Build Vocabulary:** A list of all unique words (e.g., 5,000 words) from the *entire* training dataset is created.
2. **Count:** For each document (message), count how many times each word from the vocabulary appears.

Example:

- **Vocabulary:** `[call, free, now, you, win]`
- **Message:** `"call you now, win free free"`
- **BoW Vector:** `[1, 2, 1, 1, 1]`

The “Bag”: This method is called a “bag” of words because it **loses all information about word order and grammar.**

Vectorization Method 2: TF-IDF

Problem with BoW: Very common words (like ‘you’ or ‘the’) will have high counts but aren’t very *discriminative*.

TF-IDF is a smarter weighting scheme that finds words that are *important to a specific document* but *rare in all other documents*.

The score is a product of two metrics:

1. **Term Frequency (TF):** How often a word appears in *one document*. (This is just like BoW).
2. **Inverse Document Frequency (IDF):** A measure of how rare a word is across the *entire dataset*.

Understanding TF-IDF

$TF(t, d)$ **= (Term Frequency)**

- How frequently does term `t` appear in document d?
- A word that appears 10 times in a document is probably more important than a word that appears once.

$IDF(t)$ **= (Inverse Document Frequency)**

- The “uniqueness” or “informativeness” of a term `t`.
- $IDF(t) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents with term } t}\right)$
- **Common words** (like ‘the’): The denominator is large, so the `log` is small. (Low importance).
- **Rare/Spammy words** (like ‘viagra’): The denominator is small, so the `log` is large. (High importance).

Final Score

$$TF - IDF(t, d) = TF(t, d) \times IDF(t)$$

A word gets a **high TF-IDF score** if it is **frequent in one document** (high TF) but **rare in all other documents** (high IDF). This is the perfect signal for classification.

Advanced Topic: Text Embeddings

The Problem with BoW & TF-IDF:

- These methods treat words as isolated, independent items.
- They have no concept of **meaning** or **semantics**.
- To a TF-IDF model, the words “cash,” “money,” and “currency” are as different as “cat” and “rocket.”
- It completely misses the context: “free money” and “complimentary cash” are treated as 100% different.

The Solution: Text Embeddings

- An embedding is a **dense vector** (a list of 50-300 numbers) that represents the *semantic meaning* of a word or sentence.
- These vectors are learned by neural networks (e.g., Word2Vec, GloVe, BERT) that study how words are used in context across billions of documents.

How Embeddings Capture Meaning

In an embedding's vector space, **words with similar meanings are placed close together.**

- The vector for "money" will be very close to the vector for "cash."
- The vector for "win" will be close to "winner" and "lottery."
- This space can even capture *relationships*: $vector('King') - vector('Man') + vector('Woman') \approx vector('Queen')$

Dense vs. Sparse:

- **BoW/TF-IDF** are **sparse vectors**: thousands of dimensions, almost all are zeros.
    - `[0, 0, 1, 0, 0, 0, 2, 0, 0, ..., 0]`
- **Embeddings** are **dense vectors**: all dimensions have a value.
    - `[0.12, -0.45, 0.98, 0.04, ..., -0.22]`

Embeddings for Spam Detection

Instead of feeding a model a sparse vector of word counts, we feed it a **dense vector of meaning**.

1. **Semantic Understanding:** The model now *understands* that “free cash” and “complimentary money” are very similar. It can generalize from one phrase to the other, even if it has never seen “complimentary money” before.
2. **Contextual Clues:** The model learns that spammy words (“viagra,” “urgent,” “winner,” “prize”) all live in a similar “neighborhood” of the vector space. This creates a much stronger signal for the classifier.

3. **Feature Representation:** We can represent an entire message by simply **averaging the embedding vectors** of all its words. This single, dense vector becomes a rich, semantic “fingerprint” of the message, which is then fed to the classifier.

This approach is much more powerful than just counting words.

Step 3: Model Training

After vectorization, we have a large numerical matrix (e.g., 4000 messages x 5000 features). Now we can train a classifier.

Train-Test Split

This is a **CRITICAL** step. We must evaluate our model on data it has **never seen before**.

1. **Training Set (e.g., 80%):** The model “learns” the patterns from this data. The X is the TF-IDF matrix, and the y is the `[ham, spam]` label.
2. **Test Set (e.g., 20%):** We hide this data. We only use it *once* at the very end to get an honest, unbiased score of our model’s performance in the real world.

Classification Models

We can use any binary classifier. Two are very common and effective for text:

1. **Naive Bayes (Multinomial)**
    - A **probabilistic** model based on Bayes' Theorem.
    - It calculates the probability of a message being "spam" *given* the words it contains: $P(\text{Spam}|\text{words})$
    - It's **"Naive"** because it makes a strong, incorrect assumption: that all words (features) are **independent** of each other. (It assumes 'free' and 'cash' are unrelated).
    - Despite this "naive" assumption, it works **extremely well** and **very fast** for text classification.

2. **Logistic Regression**
    - A **linear** model that finds an optimal “decision boundary” to separate the two classes.
    - It’s a very stable, reliable, and highly interpretable model.
    - It’s often a go-to benchmark for any binary classification task.

Step 4: Model Evaluation

How do we know if our model is any good? *We use the **Test Set** (the hidden data) to make predictions and compare them to the true labels.*

The Confusion Matrix

This is the most important tool for evaluation. It's a 2x2 table that shows *where* the model made mistakes.

	Predicted: HAM	**Predicted: SPAM**
Actual: HAM	**True Negative (TN)** (Correctly ignored ham)	**False Positive (FP)** (Real email sent to spam)
Actual: SPAM	**False Negative (FN)** (Spam got into inbox)	**True Positive (TP)** (Correctly caught spam)

Key Evaluation Metrics

Accuracy

- $Accuracy = \frac{TP+TN}{\textbf{Total}}$
- **What it means:** "What percentage of *all* predictions were correct?"
- **BE CAREFUL:** Accuracy is **misleading** if the dataset is **imbalanced**. If 99% of emails are Ham, a model that *only* predicts "Ham" has 99% accuracy but is useless.

Precision vs. Recall (The most important metrics)

Precision

- $Precision = \frac{TP}{TP+FP}$
- **What it means:** “Of all the emails the model *predicted* as SPAM, how many were *actually* SPAM?”
- **A high-precision model** has very few False Positives (FPs).
- **Why it matters:** This is the “user trust” metric. A low-precision model (many FPs) sends your boss’s important email to the spam folder, making users angry.

Recall

- $Recall = \frac{TP}{TP+FN}$
- **What it means:** “Of all the *actual* SPAM that exists, how many did the model *catch*?”
- **A high-recall model** has very few False Negatives (FNs).
- **Why it matters:** This is the “effectiveness” metric. A low-recall model (many FNs) lets spam into your inbox, failing at its job.

The Precision/Recall Trade-off

You can’t usually have perfect Precision and Recall at the same time.

- **Want higher Precision?** Be more “cautious.” Only flag emails you are 100% sure are spam. (This will lower your Recall, as some borderline spam gets through).
- **Want higher Recall?** Be more “aggressive.” Flag anything that looks even a little bit like spam. (This will lower your Precision, as some real emails get flagged).

The balance you choose depends on your business goal. For spam detection, **Precision is often more important** than Recall, as users hate False Positives (losing real email) more than they hate False Negatives (seeing some spam).

F1-Score

- The **harmonic mean** of Precision and Recall.
- $F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$
- A single number that balances both metrics. It’s a great way to compare two models at a glance.

Summary

1. **The Goal:** A **binary classification** task (Ham vs. Spam).
2. **Pre-processing:** We clean raw text by **tokenizing**, removing **stopwords**, and normalizing words using **Stemming** (fast/crude) or **Lemmatization** (accurate/slow).
3. **Feature Extraction:** We must convert text to numbers.
    - **Bag of Words (BoW):** Simple word counts.
    - **TF-IDF:** A powerful weighting scheme that finds “important” words (high TF, high IDF).

4. **Modeling:** We use a **Train-Test Split** to train and evaluate a model.
    - **Naive Bayes:** A fast, probabilistic model that works well for text.
    - **Logistic Regression:** A stable, linear benchmark model.
5. **Evaluation:** We measure performance on the **Test Set**.
    - **Confusion Matrix:** Shows *where* the model made errors (TP, FP, FN, TN).
    - **Precision:** Is the model trustworthy? (Avoids **False Positives**).
    - **Recall:** Is the model effective? (Avoids **False Negatives**).
    - **F1-Score:** A single metric balancing both Precision and Recall.