

**FATİH
SULTAN
MEHMET**
VAKIF ÜNİVERSİTESİ

PROJECT REPORT: SPAM DETECTION AND TEXT CLASSIFICATION USING MACHINE LEARNING AND DEEP LEARNING

Student Name & ID:

Feride Uçum – 2221251022

Furkan Emre Karçıkar- 2221251007

Semih Alev -2221251010

ABSTRACT

In this project, a robust system for binary text classification (Spam vs. Ham) was developed and analyzed. The study explores the transition from classical Machine Learning algorithms (Logistic Regression, Decision Trees) to modern Deep Learning architectures (Multi-Layer Perceptrons, Bi-LSTM). A significant portion of the work focused on Preprocessing, Feature Engineering (extracting domain-specific features), and Overfitting Prevention mechanisms such as Early Stopping, Dropout, and L2 Regularization. The results demonstrate that the MLP model utilizing TF-IDF achieved the highest accuracy (93%), outperforming complex sequential models for this specific dataset.

1. INTRODUCTION

Unsolicited commercial communication, commonly known as "spam," poses a significant threat to user security and platform integrity. Traditional keyword-based filtering is often insufficient against evolving spam tactics.

The objective of this project is to design a classifier that can distinguish between legitimate ("Ham") and malicious ("Spam") messages with high accuracy. The project implements a pipeline that includes:

- 1. Data Cleaning:** Handling noise in raw text.
- 2. Feature Extraction:** Utilizing both statistical (TF-IDF) and semantic (Word2Vec) representations.
- 3. Modeling:** Benchmarking four distinct architectures ranging from interpretable linear models to complex recurrent neural networks.
- 4. Optimization:** Implementing rigorous strategies to prevent overfitting and ensure model generalization.

2. DATA COLLECTION

2.1 Data Source and Collected Size

In this project, the dataset was collected from **YouTube video comments** using Python-based web scraping/data extraction scripts. Comments were programmatically retrieved via the youtube_comment_downloader library, and the dataset was organized into two classes:

- **Normal (label=0):** comments that do not carry spam-like signals
- **Spam (label=1):** comments that contain typical spam patterns (links/contact/CTA, etc.)

Collection summary (actual):

- **Normal URLs (videos):** 15
- **Spam URLs (videos):** 42
- **Normal comments collected:** 1170
- **Spam comments collected:** 1169
- **Total collected:** 2339 comments

The intended goal was 1500 per class; however, due to the applied filters and the available comment content under the given URL lists, the final collected dataset size is **1170 normal + 1169 spam**.

2.2 Collecting Normal Comments (label=0)

Target Size and Buffer. The normal collection script targets TARGET_NORMAL = 1500. To compensate for filtering losses, it uses a buffer: NORMAL_BUFFER = int(TARGET_NORMAL * 1.15) (1725). After collection, the results are deduplicated and truncated using head(TARGET_NORMAL) when enough samples are available. In our run, the script produced **1170** valid normal comments after filtering and deduplication.

Round-based Scanning. Normal comments are collected with a two-stage strategy:

1. **FAST PASS:** For each video, the script scans the first FIRST_PASS_SCAN = 100 comments.
2. **DEEP PASS:** The script continues scanning with step size DEEP_STEP_SCAN = 2500 until the buffer is filled, with a per-video cap MAX_TOTAL_SCAN_PER_VIDEO = 50000.

This strategy aims to first gather clean samples across multiple videos, then perform deeper scanning if needed.

English-only Filtering. The script keeps only English comments (EN_ONLY = True). The is_english_strict() function removes leading @mentions, strips non-letter characters, applies a whitelist for very short English comments, and uses langdetect.detect() for longer texts. Only comments detected as "en" are accepted.

Spam Exclusion During Normal Collection. To prevent spam from entering the normal class, the script drops comments flagged by is_spam_like(), which checks for strong spam signals such as URLs/shorteners, contact links (e.g., Telegram/WhatsApp/Discord), email/phone patterns, self-promotion phrases, and CTA phrases combined with platform/handle/link signals, as well as giveaway/scam/store keywords when combined with links.

Deduplication. Multiple dedup checks are used: comment ID (cid) deduplication, hash-based text deduplication (sha1), and author-text key filtering (author||text) to avoid repeated identical messages by the same author.

Resume and Safe Stop. The script supports resuming via checkpoint_normal_rounds.json and safe termination via STOP.txt (saving state before exit).

Outputs. Normal collection generates normal_only_en.csv, dataset_only_normal_en.csv, and per_video_counts_normal_only.csv.

2.3 Collecting Spam Comments (*label=1*)

Target Size and Constraints. Spam collection targets TARGET_SPAM = 1500 with no extra buffer (SPAM_BUFFER = TARGET_SPAM). To avoid overrepresenting “URL-only” spam, the script limits that subtype: URL_ONLY_CAP = ceil(TARGET_SPAM * 0.08) (120). From **42 spam URLs**, the script collected **1169** valid spam comments.

English-like Filtering (loose). Spam language filtering is intentionally looser (EN_ONLY_SPAM = True). The is_english_like_spam() function accepts texts with ASCII ratio ≥ 0.70 and at least one Latin letter.

Spam Detection with Reason Labels. Spam is detected using is_spam_with_reason(), which also assigns a reason label (spam_reason) such as contact_link, contact_info, self_promo, impersonation, url_only, multi_url, contact_cta, scam_kw_with_link,

`short_linkish`, `repeated_chars_link`, and `repeat_same_comment` (same author repeating the same text).

Scanning Strategy (Recent + Popular). The script scans comments sorted by `SORT_BY_RECENT` (up to `FAST_RECENT_CAP_SPAM` = 3500) and, if supported, `SORT_BY_POPULAR` (up to `FAST_POPULAR_CAP_SPAM` = 2000). If the target is not reached, deep scanning is enabled (`DO_DEEP_SPAM` = True) with `DEEP_CAP` = 20000.

Deduplication for Spam. Full text deduplication is disabled by default (`DEDUP_SPAM_BY_TEXT` = False), but repetitions are limited (`SPAM_TEXT_MAX` = 3). Duplicate `cid` values are blocked.

Resume and Safe Stop. The script supports resuming via `checkpoint_state_spam.json` and safe termination via `STOP.txt`.

Outputs. Spam collection generates `spam_only_en_1500.csv`, `per_video_spam_counts.csv`, and includes `spam_reason` for analysis.

2.4 Final Dataset Construction

After collecting normal and spam comments with separate scripts, the final binary classification dataset is formed by combining the two outputs. Normal samples use `label=0` and spam samples use `label=1`. The resulting dataset size in our run is **2339** labeled comments in total (**1170 normal + 1169 spam**).

3. DATA PREPROCESSING AND FEATURE ENGINEERING

Raw text data is unstructured and noisy. To convert it into a machine-readable format, a comprehensive pipeline was implemented in `processing.py`.

3.1. Domain-Specific Feature Engineering

Before cleaning the text, specific structural features were extracted based on domain knowledge regarding spam behavior. Spam messages often exhibit "aggressive" formatting. Using the `extract_custom_features` function, the following numerical features were generated:

- **Caps Ratio:** The ratio of uppercase letters to the total length. High values often indicate "shouting" or urgency, typical of spam.
- **Exclamation Count:** The frequency of exclamation marks (!), often used to attract attention.
- **Punctuation Ratio:** The density of punctuation characters.
- **Word Count:** The total number of words, used to analyze length distribution.

3.2. Text Cleaning Pipeline

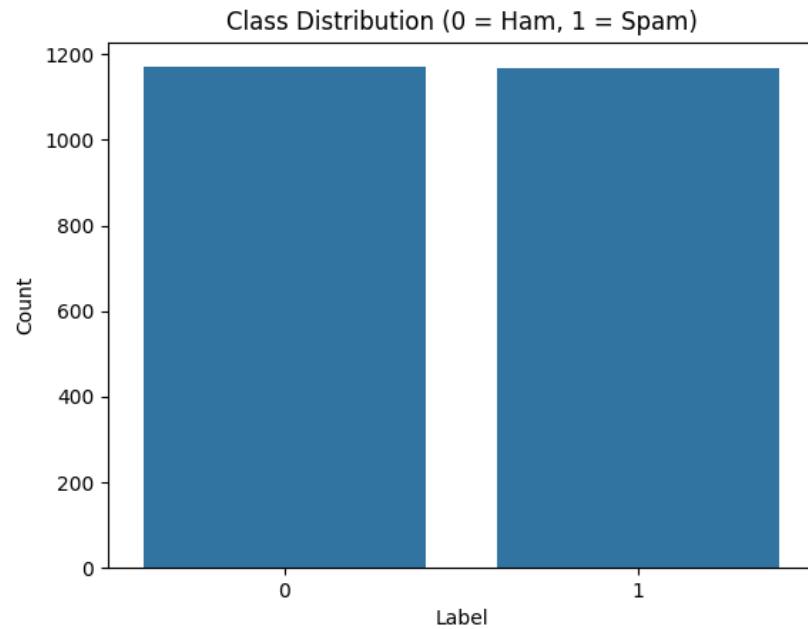
The text was normalized using the NLTK library to reduce dimensionality:

- **Noise Removal:** HTML tags and URLs were removed via Regular Expressions (RegEx) to focus on content.
- **Normalization:** All text was converted to lowercase to treat "Free" and "free" as the same token.
- **Stopwords Removal with Exceptions:** Common stopwords were removed, but critical keywords often found in spam (e.g., "you", "win", "free", "offer") were explicitly preserved to avoid information loss.
- **Lemmatization:** Words were reduced to their root forms (e.g., "winning" "win") using the `WordNetLemmatizer`.

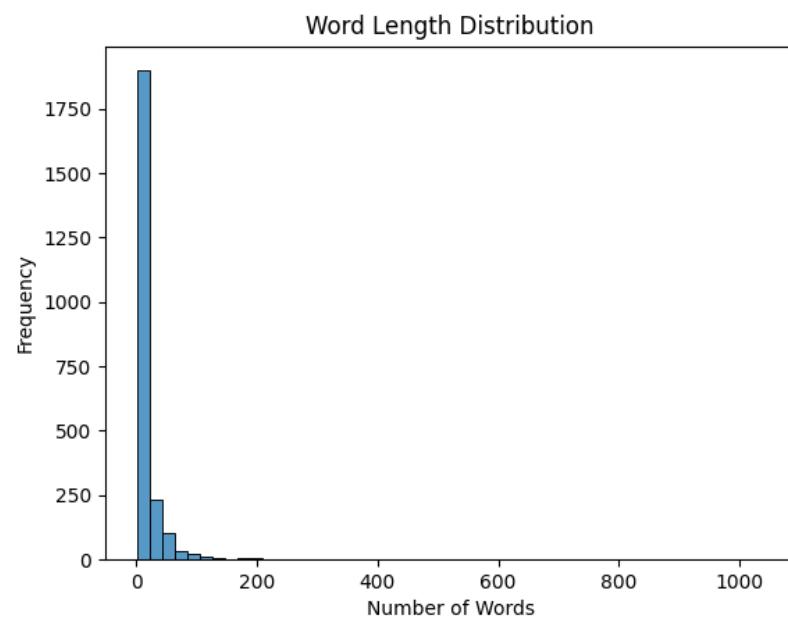
4. EXPLORATORY DATA ANALYSIS (EDA)

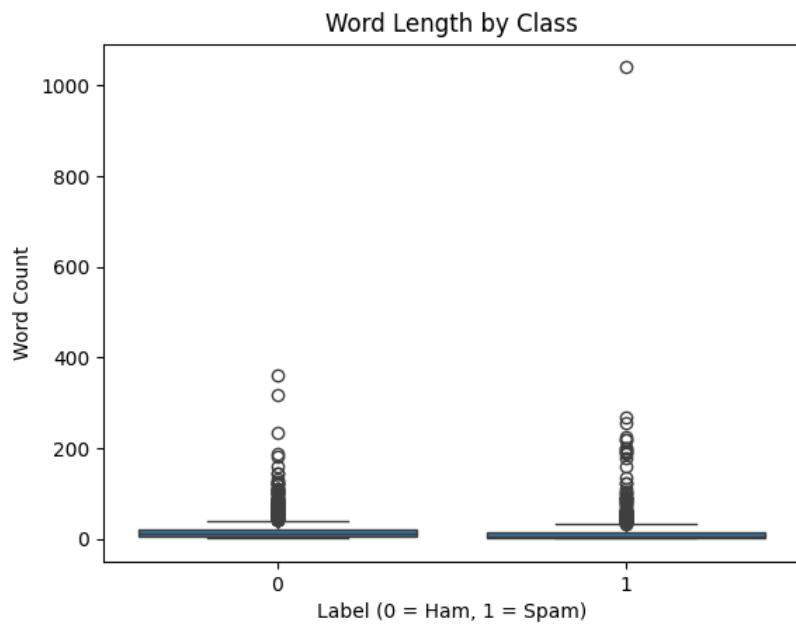
An exploratory analysis was conducted using EDA.py to understand the dataset's characteristics:

- **Class Distribution:** The balance between Spam and Ham classes was visualized to check for dataset imbalance.



- **Length Analysis:** Boxplots and histograms were generated to compare the word counts of both classes.



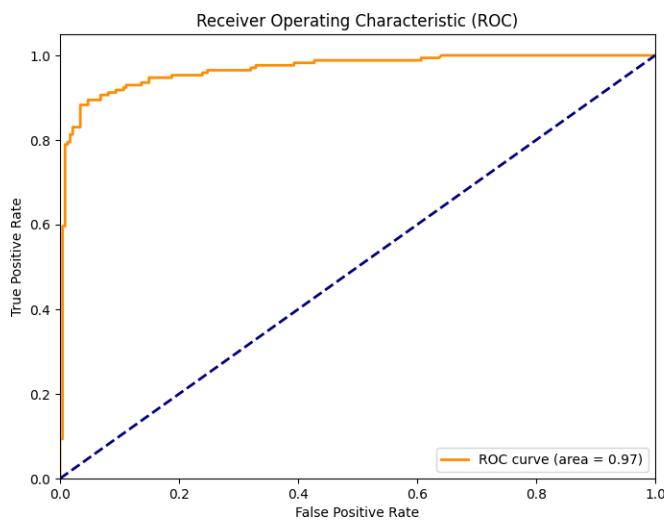
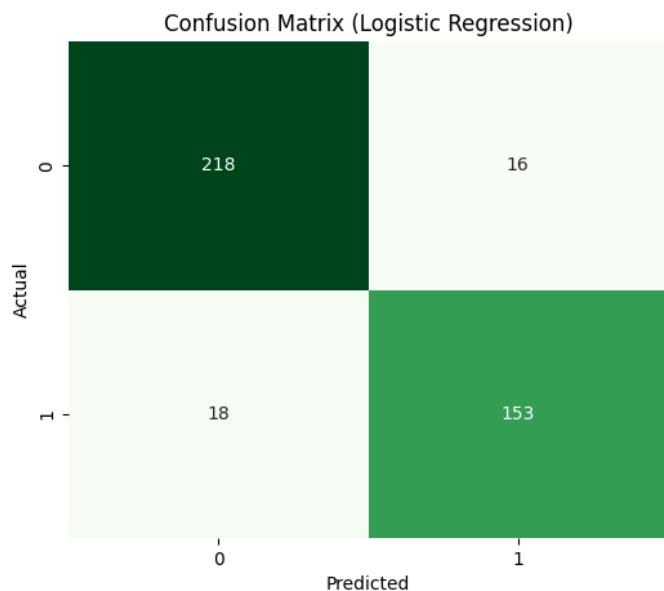


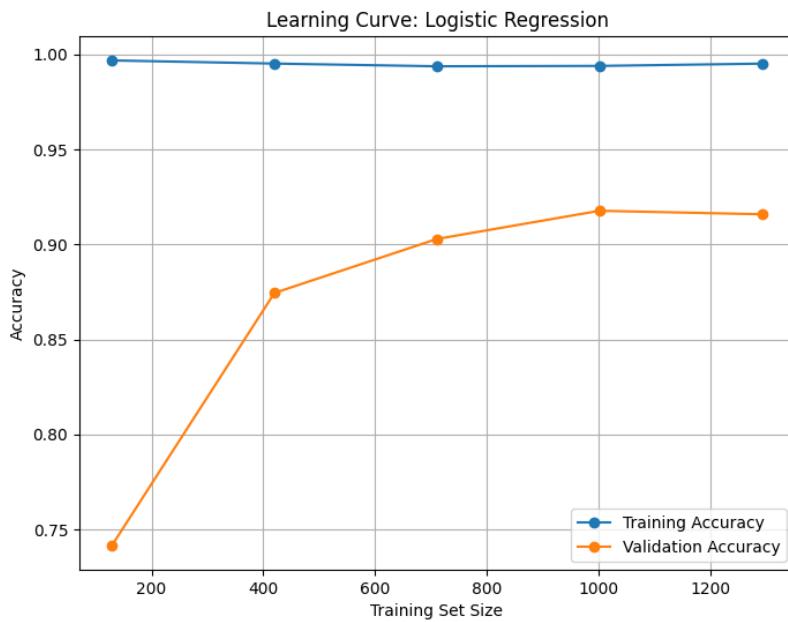
- **N-gram Analysis:** The most frequent words for both classes were identified. Unique tokens such as "call", "urgent", and "cash" were observed predominantly in the Spam class.

5. METHODOLOGY AND MODEL ARCHITECTURES

5.1. Logistic Regression (Baseline Model)

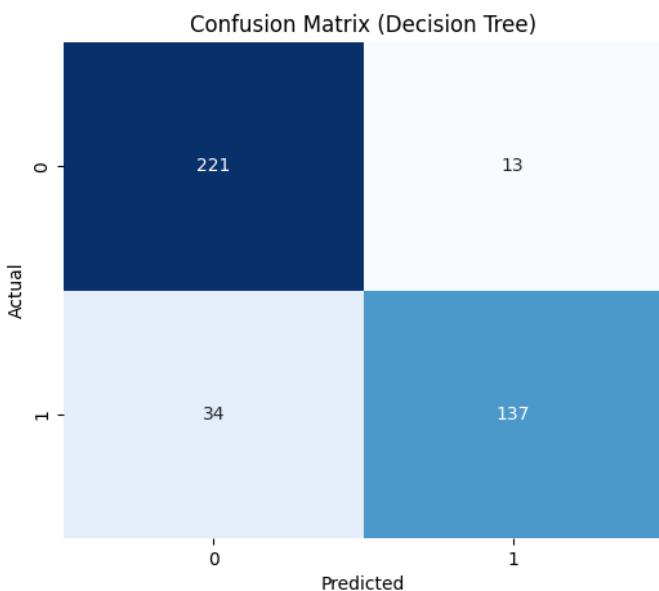
- Feature Representation: TF-IDF (Term Frequency-Inverse Document Frequency) with a maximum of 5000 features. N-grams (1,2) were used to capture local context.
- Configuration: A GridSearchCV was performed to optimize the regularization parameter C and the penalty type (L1vs L2).
- Rationale: Logistic Regression serves as a strong baseline for high-dimensional sparse data, providing interpretability and speed.

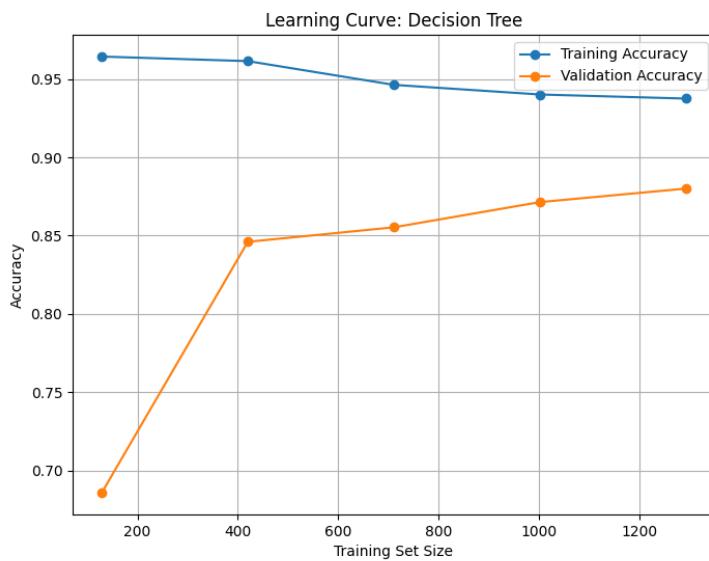




5.2. Decision Tree Classifier

- Configuration: The model was tuned to balance bias and variance. Key hyperparameters optimized included `max_depth` (to prevent growing overly complex trees) and `min_samples_split`.
- Analysis: Learning curves were generated to diagnose potential overfitting by observing the gap between training and validation scores as training size increases.

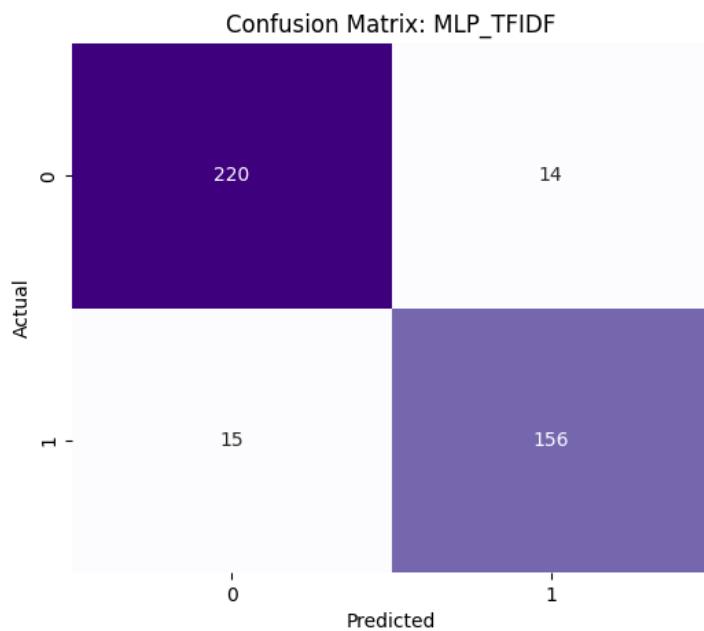


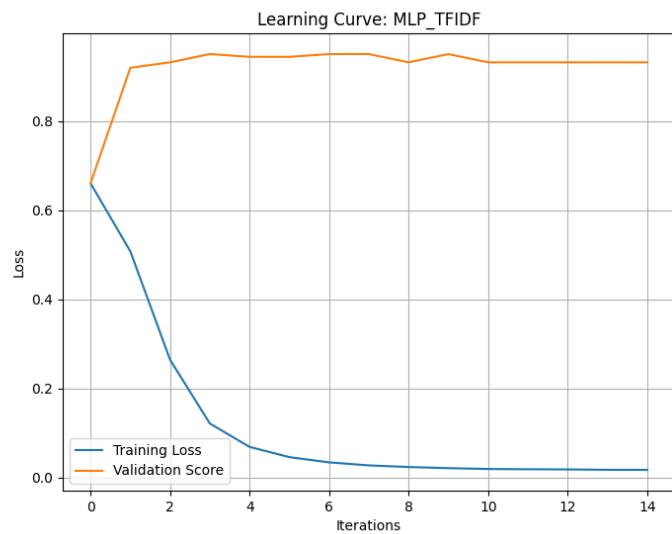


5.3. Multi-Layer Perceptron (MLP)

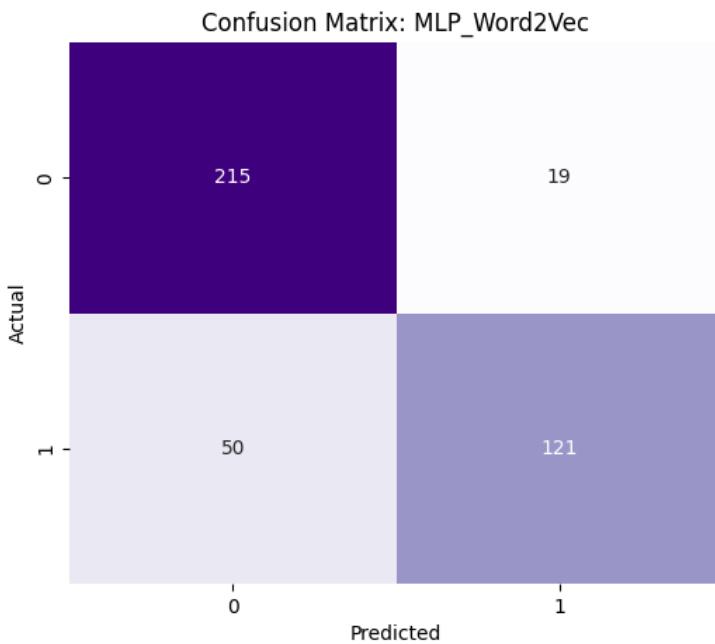
Two variations of Artificial Neural Networks were implemented in `multilayer_perceptron.py`:

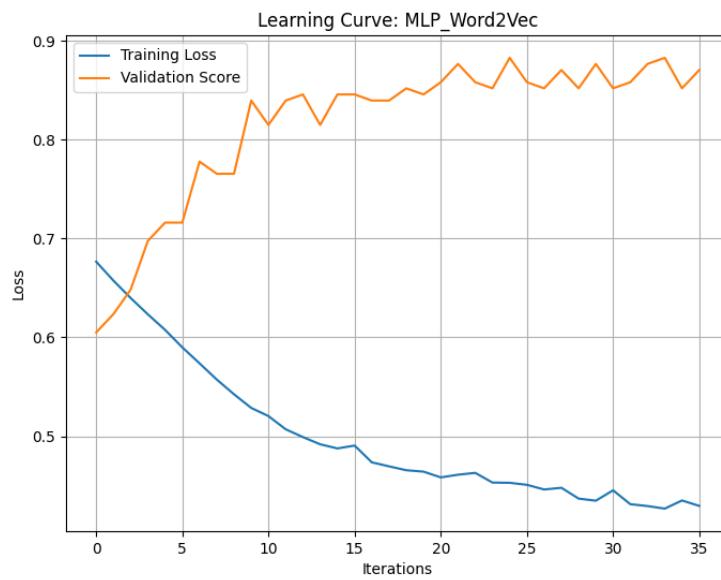
- 1. MLP + TF-IDF:** Uses the sparse TF-IDF matrix as input.





2. MLP + Word2Vec: Uses dense vector representations. A Word2Vec model was trained using the Skip-gram approach ($sg=1$) to capture semantic relationships. Sentence vectors were obtained by averaging the word vectors.



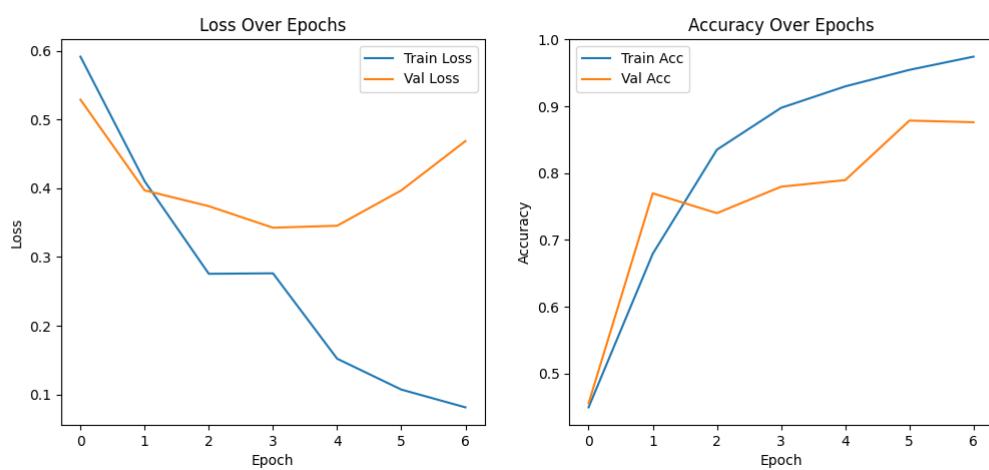
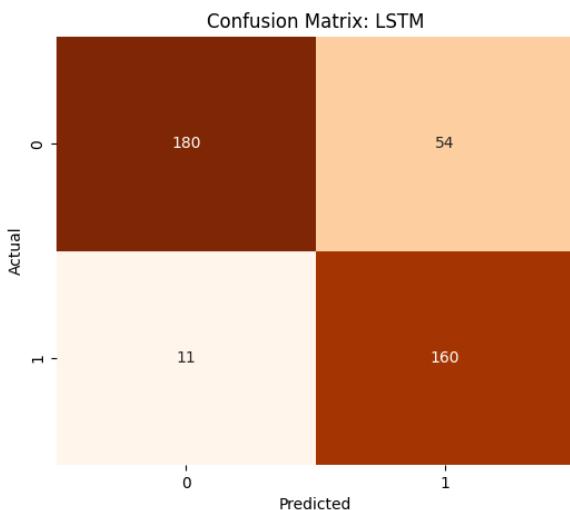


- **Architecture:** The network consists of two hidden layers (128 neurons and 64 neurons) with ReLU activation.

5.4. Recurrent Neural Network (Bi-LSTM)

To capture the sequential nature of text (where the order of words matters), a Deep Learning model was built using PyTorch in `rnn.py`.

- **Embedding Layer:** Converts integer-encoded tokens into dense vectors of size 100.
- **Bidirectional LSTM:** Unlike standard RNNs, the Bidirectional LSTM processes the text in both directions (start-to-end and end-to-start), allowing the model to understand the full context of a word.
- **Architecture:** Embedding Bi-LSTM (128 units) Dropout Fully Connected Layer.



6. OVERFITTING PREVENTION STRATEGIES

A primary focus of this project was to ensure the models generalize well to unseen data. The following specific techniques were implemented across the code files:

1. L2 Regularization (Weight Decay):

Used in Logistic Regression (via `penalty='l2'`) to penalize large weights and reduce model complexity.

2. Early Stopping:

In MLP: The `early_stopping=True` parameter was enabled in Scikit-Learn to halt training when validation scores ceased to improve.

In LSTM: A manual patience mechanism (set to 3 epochs) was implemented in the training loop. The system monitors the Validation Loss; if it does not decrease for 3 consecutive epochs, training is terminated, and the best model state is restored via `torch.save / torch.load`.

3. Dropout:

Applied in the LSTM architecture with a rate of 0.5 (50%). This randomly deactivates neurons during training, forcing the network to learn robust features rather than relying on specific pathways.

4. Cross-Validation:

A 5-Fold Cross-Validation was used during the hyperparameter tuning of the Decision Tree and Logistic Regression models to ensure statistical reliability.

7. RESULTS AND DISCUSSION

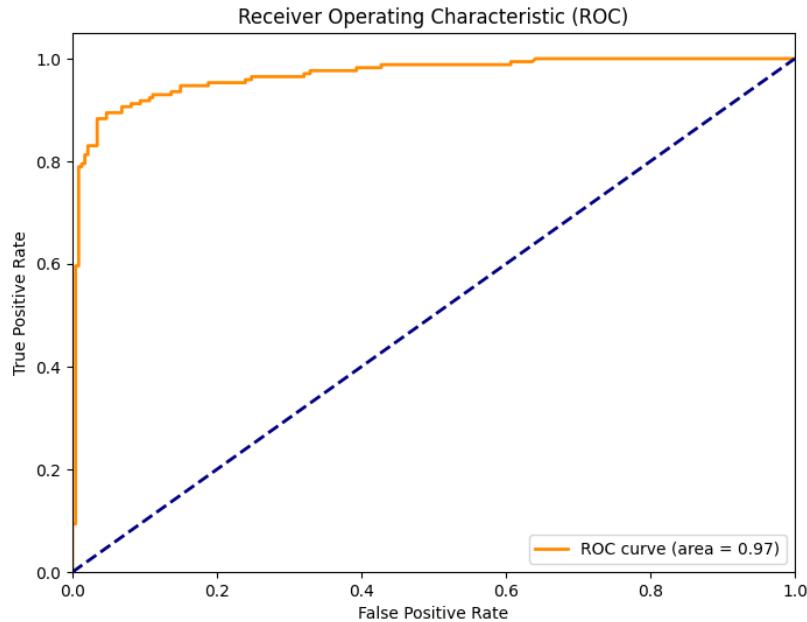
The models were evaluated on a held-out Test Set (20% of the data). The performance metrics include Accuracy, F1-Score (weighted), Precision, and Recall.

MODEL	ACCURACY	F1-SCORE	PRECISION	RECALL
Logistic Regression	0.92	0.92	0.92	0.92
Decision Tree	0.88	0.88	0.89	0.88
MLP (TF-IDF)	0.93	0.93	0.93	0.93
MLP (Word2Vec)	0.83	0.83	0.83	0.83
Bi-LSTM (RNN)	0.84	0.84	0.84	0.84

7.1. Performance Analysis

- **Best Model:** The MLP (TF-IDF) model achieved the highest performance with an accuracy of 93%. This result indicates that for this specific dataset size, the explicit presence of key spam terms (captured efficiently by TF-IDF) is a stronger predictor than the semantic context.
- **TF-IDF vs. Embeddings:** Models using TF-IDF (Logistic Regression and MLP) significantly outperformed models using embeddings (Word2Vec MLP and LSTM). This suggests that the dataset may be too small to train effective dense embeddings, or that the semantic "meaning" is less important than specific keyword triggers (e.g., "FREE", "WIN") for spam detection.
- **LSTM Evaluation:** While the LSTM achieved a lower overall accuracy (84%), it is worth noting its Recall for the Spam class (Class 1) was very high (0.94). This means the LSTM is extremely aggressive in catching spam but suffers from a lower precision (0.75), producing more false positives than the MLP.

- **Overfitting Check:** The implementation of Early Stopping was successful. The LSTM model stopped training early when validation loss plateaued, preventing the memorization of noise.



7.2 Error Analysis (Logistic Regression)

To better understand the limitations of our best-performing classical baseline, we inspected misclassified samples produced by the Logistic Regression model on the held-out test split. The model produced **34** incorrect predictions in total.

(Insert Figure 17 here: results/models/logistic_regression/confusion_matrix.png — Confusion Matrix for Logistic Regression)

7.2.1 False Positives vs. False Negatives

We decomposed the errors into:

- **False Positives (Ham → Spam): 16**
- **False Negatives (Spam → Ham): 18**

This indicates the model produces slightly more false negatives than false positives, meaning it sometimes fails to flag spam as spam. Depending on the application, this trade-off matters: false negatives reduce spam-catching capability, while false positives risk incorrectly blocking legitimate user comments.

7.2.2 Representative Misclassified Examples (with Confidence)

Below are representative misclassified examples, including the model's estimated probability of the comment being spam ($p(\text{spam})$):

False Positives (Ham predicted as Spam):

- $p(\text{spam})=0.913$ | True: 0 → Pred: 1 | must reason playlist suddenly popped recommendation
- $p(\text{spam})=0.727$ | True: 0 → Pred: 1 | apparently screwed
- $p(\text{spam})=0.904$ | True: 0 → Pred: 1 | duping people joining your discord
- $p(\text{spam})=1.000$ | True: 0 → Pred: 1 | want update
- $p(\text{spam})=0.983$ | True: 0 → Pred: 1 | indoctrination real

False Negatives (Spam predicted as Ham):

- $p(\text{spam})=0.039$ | True: 1 → Pred: 0 | follow guy thanks vanndicoot
- $p(\text{spam})=0.430$ | True: 1 → Pred: 0 | digital india job offer ... work whatsapp
- $p(\text{spam})=0.011$ | True: 1 → Pred: 0 | nice
- $p(\text{spam})=0.013$ | True: 1 → Pred: 0 | aww thank you ... checked link ... let know open

7.2.3 Observed Failure Modes

From these misclassifications, we observe several recurring patterns:

1. Very short / low-context comments

Extremely short comments (e.g., nice, want update) provide limited lexical evidence. In such cases, TF-IDF-based linear models can rely on weak token correlations and may assign overly confident probabilities, resulting in both false positives and false negatives.

2. Spam without strong “obvious” spam markers

Some spam messages do not contain explicit URLs or clearly recognizable contact patterns, leading to false negatives (e.g., follow guy thanks vanndicoot). This suggests that a portion of spam in our dataset appears “soft” or relies on subtle persuasion rather than overt links.

3. Ambiguous platform/community tokens

Tokens like *discord* can appear in both legitimate discussion and spam promotion. This ambiguity can increase false positives, as seen in duping people joining your discord.

8. CONCLUSION

This project successfully demonstrated the application of NLP techniques for spam detection. While Deep Learning models like LSTM offer theoretical advantages in understanding context, this study proves that **Linear Models (Logistic Regression)** and **Simple Neural Networks (MLP)** combined with **TF-IDF** can provide superior performance on specific text classification tasks where keyword frequency is the dominant feature. The robust preprocessing and overfitting prevention strategies ensured that the final model (MLP) is both accurate and generalizable.