VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANASANGAMA,BELAGAVI-590014

Technical Seminar Report
on

"TEXT CLASSIFICATION AND SPAM DETECTION"

Submitted for partial fulfillment of requirement for the award of Degree of

Bachelor of Engineering

in

Computer Science & Engineering (Data Science)

Submitted by

MEHEK A	**[1RL22CD028]**
NASREEN T S	**[1RL22CD032]**
NAVJOT KAUR	**[1RL22CD033]**
SAYADA RUQAYYA	**[1RL22CD047]**

Under the guidance of

Mrs B C Hemapriya
Assistant Professor

Department of Computer Science & Engineering (Data Science)
R. L. JALAPPA INSTITUTE OF TECHNOLOGY
Doddaballapur, Bangalore Rural Dist-561 203

2024-25

Sri Devaraj Urs Educational Trust (R.)
R. L. JALAPPA INSTITUTE OF TECHNOLOGY
(Approved by AICTE, New Delhi & Affiliated to VTU, Belagavi)
Kodigehalli, Doddaballapur- 561 203

Department of Computer Science & Engineering (Data Science)

CERTIFICATE

Certified that the Technical Seminar work entitled **"Text Classification and spam detectiom"** **carried out by Mehek A [1RL22CD028], Nasreen T S [1RL22CD032], Navjot Kaur [RL22CD033], Sayada Ruqayya [1RL22CD047]** are bonafide students of R.L.Jalappa Institute of Technology, in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visveswaraya Technological University, Belagavi during the year 2024-25. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the department library. The Technical Seminar report has been approved as it satisfies the academic requirements in respect of Technical Seminar work prescribed for the said degree.

Mrs. B C Hemapriya **Assistant Professor** **Dept of CSE,RLJIT**	**Dr. Mrutyunjaya M S** **Associate Professor & HOD** **Dept of CSE(Data Science), RlJIT**	**Dr. P. Vijayakarthik** **Principal** **RLJIT**

Name of the Examiners

1. _____ _____

2. _____ _____

ABSTRACT

In today's digital world, people are constantly exposed to massive amounts of text—emails, messages, social media posts—many of which include spam or unwanted content. This project focuses on using Natural Language Processing (NLP) and Machine Learning to automatically detect and classify spam messages. Our goal was to build an intelligent system that can distinguish between spam and genuine (ham) messages in real time with high accuracy.

We began by collecting data from well-known sources like the SMS Spam Collection and Enron Email Dataset. After cleaning and preprocessing the data—removing noise, stop words, and standardizing the text—we extracted features using methods like TF-IDF, n-grams, and word embeddings. We trained multiple models including Naive Bayes, SVM, and deep learning models like BERT and DistilBERT.

The results were promising. While traditional models like Naive Bayes achieved decent accuracy, the DistilBERT model outperformed them all with a 93% accuracy rate, proving how effective transformer-based models are for text classification tasks. We also implemented real-time testing and explored ethical considerations like privacy and model transparency.

This project demonstrates how machine learning can be used to improve digital communication by filtering out spam and keeping inboxes clean, safe, and efficient.

ACKNOWLEDGEMENT

The completion of seminar presentation brings with a sense of satisfaction, but it never completes without thanking the persons responsible for its successful completion.

I would like to express our profound grateful to His Divine Soul **Sri R.L. Jalappa founder**, of Sri Devaraj Urs Educational Trust, Kolar, for providing us an opportunity to complete our academics in this esteemed institution.

I extend my deep sense of sincere gratitude to **Dr. Vijayakarthik P, Principal**, for providing me an opportunity to do my course.

I express my heartful sincere gratitude to **Dr. Mrutyunjaya M S, Associate Professor & Head**, Department of Computer Science and Engineering(Data Science), for his valuable suggestions and support.

I express our truthful thanks to my guide,**Mrs. B C Hemapriya, Assistant Professor**, Department of Computer Science and Engineering, for his valuable guidance, suggestions and cheerful encouragement throughout the internship.

Finally, I would like to thank all the teaching and non-teaching members of Department of Computer Science and Engineering for their support.

Mehek A [1RL22CD028]

Nasreen T S [1RL22CD032]

Navjot Kaur [1RL22CD033]

Sayada Ruqayya [1RL22CD047]

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER-1

INTRODUCTION

In the increasingly vast virtual world, we are surrounded all the time by huge amounts of text data — emails, social media updates, product reviews, articles, and so on. Interpreting this unstructured data is a significant problem in the domain of data science. Text classification is one of the fundamental methods that assist in dealing with and analyzing such data. Text classification is the task of classifying text into categories or labels based on what the text says. It has a large impact in many applications, ranging from spam filtering and sentiment analysis to topic classification and language identification.

Text classification is a natural language processing (NLP) subtask, and NLP is an artificial intelligence (AI) branch concerned with the relationship between computers and human language. In text categorization, machine learning models learn from a labeled data set — in which each piece of text is labeled with its respective category — and apply this knowledge to categorize new, unseen text. For instance, a system may be trained on a data set of emails classified as "spam" or "ham" (not spam) and subsequently employed to automatically sort incoming emails.

One of the most ubiquitous and useful applications of text classification is spam filtering. Spam is unwanted and irrelevant messages that are typically sent in bulk, frequently for malicious purposes like phishing, fraud, or advertising. Spam emails can clog inboxes, lower productivity, and even result in data breaches if not effectively caught and filtered. It is for this reason that having an effective spam filtering system is an important application in digital communication and cybersecurity.

Spam filtering is generally considered as a binary categorization problem where messages are tagged as spam or ham. It is trained to identify some of the patterns, keywords, and behaviors characteristic of spam messages. For example, the occurrence of too many hyperlinks, particular promotional sentences, or suspicious sender addresses may label a message as spam. Today's spam filters are more complex than simple keyword identification and instead employ machine learning algorithms that monitor the structure and semantics of a message.

Machine learning models that are widely applied to text classification and spam filtering are Naive Bayes, Support Vector Machines (SVM), Decision Trees, Random Forests, and more recently, deep learning models like Recurrent Neural Networks (RNNs) and Transformers. Of these, the Naive Bayes classifier is especially favored in spam filtering because it is simple, fast, and efficient in processing text data.

The text data employed in spam detection needs to go through various preprocessing steps prior to being employed for model training. These involve tokenization (breaking text into words or tokens), stopword removal (normal words such as "and," "the," "is"), stemming or lemmatization (reducing words to their base form), and vectorization (converting text to numerical features). These popular vectorization methods are Bag of Words (BoW), TF-IDF (Term Frequency–Inverse Document Frequency), and word embeddings such as Word2Vec or GloVe.

The performance of a spam filtering system is measured with accuracy, precision, recall, F1-score, and confusion matrix. An ideal model should be precise (not many false positives, i.e., valid emails incorrectly labeled as spam) and recall (not many false negatives, i.e., spam messages that pass through the filter). The tradeoff between these metrics is necessary to make sure that the filter for spam is effective and not excessively arbitrary.

Spam messages have evolved over time and now employ techniques of obfuscation and social engineering to evade detection systems. Therefore, contemporary spam detection models need to be adaptable and constantly updated to cope with changing threats. This is where AI and real-time learning systems are being incorporated in spam filters to make them more functional and responsive.

Text classification and spam detection are not only applicable for emails. They are applied far and wide through messaging platforms, comment fields, customer support services, and even mobile SMS software. Telecom organizations, for example, employ spam detection software in order to suppress scam messages as well as unwanted promotional messages. Social media also employs content classification to identify spam comments or duplicate accounts advertising harmful links.

One of the key challenges in constructing spam detection systems is having access to high-quality labeled data. It is crucial to create and update a dataset that accurately represents real-world situations for training effective models. Public datasets such as the SMS Spam

Collection, Enron Email Dataset, and SpamAssassin Public Corpus are commonly employed in research and development initiatives. These datasets facilitate the benchmarking of various models and enhancing classification methods.

Ethical considerations also come into play in spam filtering and text categorization. Though the intent is to exclude injurious content, false positives are a risk, where legitimate exchange is blocked or marked as spam. Additionally, privacy issues need to be dealt with, particularly if it involves analyzing user data. Transparency, fairness, and respect for user privacy are what is needed in such systems of spam detection.

Over the past few years, deep learning and natural language understanding (NLU) have further increased the power of text classification systems. Models such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformers) are employed to grasp context, intent, and semantics better. These models can classify spam more accurately, particularly when messages are designed to be as close to natural communication as possible.

To develop a spam detection system as a mini project, one generally has to go through a sequence of steps: gathering and preparing a labeled dataset, text data preprocessing, choosing suitable features, selecting and training a machine learning model, and lastly testing its performance. Python, scikit-learn, NLTK, Pandas, and TensorFlow/Keras are some tools and libraries available that can be utilized to create and test such a system.

In summary, spam detection and text classification are influential methods in artificial intelligence that play a fundamental role in filtering, structuring, and safeguarding electronic communication. Through the use of machine learning and NLP, these systems have the ability to recognize and block unwanted material in an intelligent manner, boost user experience, and protect sensitive data. With electronic communication rapidly evolving, the need for robust and smart text classification systems will keep increasing. A mini project in spam detection not only gives students exposure to fundamental concepts of machine learning and NLP but also hands-on practice in tackling real problems with real-world relevance.

CHAPTER-2

LITERATURE SURVEY

Spam detection has become increasingly important with the rise of digital communication. From emails to social media, unwanted and harmful content continues to affect users daily. To tackle this, researchers have explored various methods using Natural Language Processing (NLP), machine learning, and deep learning. This literature survey reviews several papers that focus on improving spam detection systems. These studies apply different models—from traditional techniques like Naive Bayes to advanced approaches like BERT and DistilBERT—each offering valuable insights into accuracy, performance, and real-time application. The goal is to build smarter, faster, and more reliable spam classification systems.

SL No	Authors	Title Of The Paper	Contribution	Limitations
1	• Aditya Srivastava • Pawan Singh	Spam Detection Using NLP	• Developed a complete spam detection pipeline using NLP, from data collection to model evaluation. • Applied NLP techniques to extract meaningful features from text. • Compared various machine learning	• Struggles with data imbalance and misclassificatio n. • Limited adaptability to evolving spam tactics. • Real-world

			models (e.g., Naive Bayes, SVM, Neural Networks). • Analyzed real-world challenges like data imbalance and evolving spam tactics. • Suggested future directions using advanced NLP models like BERT and real-time detection.	deployment and privacy concerns not deeply addresse d.
2	• Hadeer Ahmed • Issa Traore • Sherif Saad	Detecting Opinion Spams And Fake News Using Text Classificaton	• Used n-gram features with TF and TF-IDF for feature extraction. • Compares six ML algorithms with a best accuracy of 92%. • Presented a new large fake news dataset for research.	• Dependent on heavily abelled sets of data; performance can decrease with noisy or unlabeled real-world data. • Is restricted to text-based features, excluding multimedia content such as images or video within fake news. • N-gram and

				traditional ML approaches will not generalize well to highly advanced or context-dependent fake content. • Restricted to English-language datasets; performance on other languages is unknown.
3	• Sanaa Kaddoura • Ganesh Chandrashekaran • Daniela Elena Popescu • Jude Hemanth Duraisamy	A Systematic Literature Review On Spam Content Detection And Classification	The paper gives a thorough overview of spam detection and classification methods in social media. It discusses rule-based, machine learning, and deep learning approaches and some preprocessing and feature extraction methods such as TF-IDF and Word2Vec. It also enumerates important datasets, tools, present challenges, and	• Limited labeled data for training and testing. • Class imbalance in most datasets (excess spam or ham). • Language and geographic bias in gathered data. • Deep learning models have high resource needs. • Spammers' changing

			provides recommendations for future studies, thus constituting an effective handbook for researchers in spam detection.	strategies make it challenging to d etect
4	• G.Jain • Manisha • B.Agrawal	Spam Detection On Social Media Text	The article "Spam Detection on Social Media Text" compares five classical machine learning classifiers—Naïve Bayes, SVM, KNN, ANN, and Random Forest—for detecting spam using SMS and Twitter datasets. It uses text preprocessing (special character, stop word, and hyperlink removal) and feature extraction (Bag of Words with TF and TF-IDF).	• Only traditional ML models used, no deep learning. • Manual labeling of Twitter data may be biased. • Relies on handcrafted features (BoW), lacks semantic understanding. • Small dataset limits generalization. • Not tested in real-time scenarios.
5	• Tianrui Lui • Shaojie Li • Yushan Dong • Yuhong Mo	Spam Detection And Classification Based On Distilberst Deep Learning Algorith	The paper provides significant value by proving that the lightweight DistilBERT model is able to successfully	• Restricted Dataset Diversity • Misclassificatio n Problems • No Real-Time

	• Shuyao He	m	identify spam emails with 93% accuracy. With the synergy of deep learning and sophisticated text preprocessing, it provides a potent, efficient solution to enhancing email security and user experience—key to today's digital communication era.	Testing • Resource Limitations • Absence of Comparison • Limited Feature Engineering

Proposed Solutions to Overcome Identified Limitations

- **Data Augmentation**:
  - Use techniques like synonym replacement, back-translation, or contextual augmentation using language models (e.g., BERT) to expand the training data.
- **Semi-supervised / Self-supervised Learning**:
  - Train models using a small labeled set + large unlabeled data.
  - Use pseudo-labeling: Let the model label data and retrain iteratively.
- **Synthetic Oversampling**:
  - Techniques like **SMOTE**, **ADASYN**, or **GAN-based** text generators can create synthetic minority class samples to balance datasets.
- **Transfer Learning**:
  - Use pretrained models like BERT, RoBERTa, or GPT, which generalize better even with limited labeled data.
- **Cross-lingual / Multilingual Models**:

    - Use multilingual BERT or XLM-RoBERTa to handle geographic/language biases.
- **Crowdsourcing**:
    - Use platforms like Amazon Mechanical Turk for diverse labeling. Ensure quality via redundancy and validation heuristics.
- **Use Contextual Word Embeddings**:
    - Switch from TF-IDF/N-grams to embeddings from **BERT**, **FastText**, or **Sentence Transformers** to capture semantics and context.
- **Automated Feature Engineering Tools**:
    - Use libraries like **FeatureTools** or **AutoFeat** to create features programmatically.
- **Hybrid Models**:
    - Combine handcrafted features (BoW, n-grams) with embeddings to capture both surface-level and semantic features.
- **Topic Modeling**:
    - Use **LDA** or **BERTopic** to extract latent topics as additional features.
- **Deep Learning Architectures**:
    - Use **CNN** or **BiLSTM** models for sequence modeling.
    - Transformers like **BERT**, **DistilBERT**, or **ALBERT** fine-tuned on spam datasets can outperform classical models.
- **Ensemble Learning**:
    - Combine multiple models (e.g., NB + SVM + RF) using voting or stacking.
    - Improve generalization and reduce misclassification.
- **Class Weighting / Cost-sensitive Learning**:
    - Modify loss function to penalize false negatives more (especially in spam detection).
    - Many libraries allow class_weight='balanced'.
- **Error Analysis + Confusion Matrix Optimization**:
    - Focus on improving precision or recall based on application need.
- **Real-time Testing Framework**:
    - Build a pipeline using **Flask**, **FastAPI**, or **Streamlit** to simulate real-time predictions.
    - Benchmark latency and throughput.

- **Model Compression**:
  - Use **quantization**, **knowledge distillation**, or **ONNX** conversion to deploy deep models in resource-constrained environments.
- **Federated Learning**:
  - Train models on user devices without sending data to a server — helps with **privacy**.
- **Differential Privacy Techniques**:
  - Use differentially private noise injection during training to protect user data.
- **Edge Deployment**:
  - Export models to run on mobile devices or browsers using **TensorFlow Lite**, **ONNX**, or **TF.js**.
- **Use Balanced Metrics**:
  - Always report **Precision, Recall, F1 Score**, and **ROC-AUC**, especially in imbalanced datasets.
- **Robustness Checks**:
  - Evaluate model on **adversarial examples**, **noisy inputs**, or **new domains** to test generalization.
- **Cross-Domain Evaluation**:
  - Train on SMS, test on emails or tweets to check domain adaptability.

CHAPTER-3

METHODOLOGY

This chapter outlines the methodology adopted for the development of a spam detection system using Natural Language Processing (NLP) and Machine Learning techniques. The approach is derived from an analysis of existing literature and enhanced with proposed methods to improve performance, scalability, and contextual understanding.

The process is divided into several phases:

3.1 Data Collection

The foundation of any machine learning model lies in the quality and diversity of the data it is trained on. For this project, we focus on collecting **textual data** that clearly distinguishes between **spam** (unwanted or malicious messages) and **ham** (legitimate messages). Since creating our own labeled dataset would be time-consuming and error-prone, we rely on widely used **public datasets** that are benchmarked in many academic studies.

5.1.1 Datasets Used

- **SMS Spam Collection Dataset (UCI Repository):**
  - Contains 5,574 labeled SMS messages in English.
  - Each message is marked as either spam or ham.
  - It's a balanced dataset commonly used in text classification research.
- **Enron Email Dataset:**
  - A real-world email dataset consisting of messages sent and received by Enron employees.
  - Includes both spam and non-spam (ham) emails.
  - Provides more complex and diverse content compared to SMS data.
- **Twitter Spam Dataset (optional/advanced):**
  - Short-form content, useful for studying spam on social media platforms.
  - Offers noisy, real-time language that adds variety to the training data.

5.1.2 Data Format

Most datasets come in **CSV format**, with fields such as:

- text: The content of the message.
- label: The class — "spam" or "ham" (sometimes encoded as 1 or 0).

5.1.3 Challenges in Data Collection

- **Class Imbalance:** In most real-world data, spam is a small fraction of total messages. This could lead to biased learning where the model favors "ham."
- **Language Diversity and Noise:** Messages might include slang, emojis, abbreviations, or even code-switching (mix of languages).
- **Privacy and Licensing:** Some datasets may include sensitive information. It's crucial to work with datasets that are anonymized and publicly licensed.

To address these challenges, we explore **balancing techniques** (explained later) and choose datasets with clear licensing and ethical usage terms.

3.2 Data Preprocessing

Raw text data is often messy, inconsistent, and full of irrelevant symbols. Before we feed any data into our models, we apply a **series of preprocessing steps** to clean and normalize it. These steps are crucial because they help reduce noise and ensure that the model focuses only on meaningful patterns.

5.2.1 Cleaning and Normalizing Text

- **Lowercasing:** All characters are converted to lowercase so that "Free" and "free" are treated the same.
- **Removing Special Characters and Punctuation:** Characters like !, #, @, and & are removed unless they're contextually important (like in hashtags or emails).
- **Removing Numbers (optional):** Depending on the application, numeric digits are either removed or replaced with a token like [NUMBER].

5.2.2 Tokenization

- The text is split into smaller units called **tokens** (usually words).
- Example: "Win a FREE iPhone now" becomes ["win", "a", "free", "iphone", "now"].

5.2.3 Stop Word Removal

- Words that occur frequently but carry little meaning (like "is", "the", "and") are removed using a predefined stopword list (from NLTK or SpaCy).
- This reduces the size of the vocabulary and keeps only meaningful terms.

5.2.4 Lemmatization and Stemming

- **Stemming** removes suffixes to find the root form of a word. E.g., "running" → "run".
- **Lemmatization** is more advanced and uses grammar rules to find the correct base form. E.g., "better" → "good".
- We use **lemmatization** because it preserves context more accurately than crude stemming.

5.2.5 Removing or Replacing Entities

- **URLs** are replaced with [URL].
- **Email addresses**, **phone numbers**, and **user mentions** (@username) are removed or masked.
- **Emojis** can be removed or translated into words using emoji libraries if necessary.

5.2.6 Handling Language-Specific Noise

If the dataset contains multiple languages or informal speech:

- Use **language detection** to filter or preprocess accordingly.
- Normalize abbreviations and internet slang (e.g., “gr8” → “great”).

This step ensures a **clean and uniform dataset**, which significantly improves the performance of feature extraction and classification.

3.3 Feature Extraction

Once we've cleaned and preprocessed the text, we need to convert it into a **numerical format** that machine learning models can understand. This process is known as **feature extraction**.

5.3.1 Bag of Words (BoW)

- Represents each message by counting how many times each word appears.
- Creates a large, sparse matrix where each row is a message and each column is a word.
- Simple but effective for many ML algorithms.

5.3.2 Term Frequency–Inverse Document Frequency (TF-IDF)

- Improves on BoW by giving more importance to **unique words** that appear in fewer messages.
- Formula:
    - **TF** = frequency of word in a message
    - **IDF** = log(total messages / messages containing the word)
- Helps highlight keywords like "win", "lottery", and "urgent" that are often found in spam.

5.3.3 N-Grams

- Captures sequences of words, not just individual words.
- A bigram model for "win big prize" would extract "win big" and "big prize" as features.
- Improves detection of patterns used in spam like "click here", "limited offer", etc.

5.3.4 Word Embeddings

- Transforms words into vectors in a way that captures **meaning** and **context**.
- We use pre-trained models such as:
    - **Word2Vec** (Google)
    - **GloVe** (Stanford)

    - **FastText** (Facebook)
- These embeddings allow the model to understand relationships between words — like "free", "offer", and "discount" being related.

5.3.5 Contextual Embeddings (BERT)

- Unlike Word2Vec, BERT embeddings are **context-sensitive**. For example:
    - In “he went to the bank”, the word “bank” means different things depending on whether it’s a financial institution or a riverbank.
- We extract embeddings from **pre-trained BERT models** (like BERT-Base or DistilBERT) to represent the entire message.

These feature representations — especially when combined — form a strong foundation for training accurate and intelligent spam detection models.

3.4 Model Selection and Training

A variety of classifiers are evaluated based on literature suggestions:

Traditional Machine Learning Models:

- Multinomial Naive Bayes (MNB): Baseline model known for speed and performance on text data.
- Support Vector Machines (SVM): Used for its robustness in high-dimensional space.
- Random Forest: For ensemble-based classification with feature importance analysis.
- Logistic Regression: Chosen for its simplicity and interpretability.

Deep Learning and Transformer Models:

- LSTM (Long Short-Term Memory): For sequence-based classification.
- CNN for Text: To capture n-gram patterns via filters.
- DistilBERT and BERT Fine-Tuning: Advanced contextual spam detection with transfer learning.

The dataset is split into 80% training and 20% testing, with stratified sampling to maintain class balance. A portion (10%) of training data is used for validation.

3.5 Evaluation Metrics

To measure performance, the following metrics are computed:

- Accuracy – Overall correct predictions.
- Precision – Correctness of positive predictions (spam).
- Recall – Ability to find all spam messages.
- F1-Score – Harmonic mean of Precision and Recall.
- ROC-AUC Score – Evaluates classifier across all thresholds.
- Confusion Matrix – Visualizes classification outcomes: TP, TN, FP, FN.

These metrics help in tuning trade-offs between misclassifying spam and flagging valid messages.

3.6 Proposed Enhancements and Methods

To improve the accuracy, adaptability, and overall performance of our spam detection system, we propose a set of advanced methods that go beyond traditional approaches. These methods are inspired by what we learned from the literature survey and also reflect the current trends in Natural Language Processing (NLP) and Machine Learning (ML). Our goal is to build a system that's not only effective in detecting spam but also flexible enough to handle real-world challenges like class imbalance, privacy, and changing spam tactics.

3.6.1 Combining Different Types of Features

Instead of relying on just one way to represent the text (like word counts or TF-IDF), we propose using a combination of feature extraction techniques. This helps our model understand the text at different levels — both surface-level keywords and deeper meanings.

- TF-IDF (Term Frequency-Inverse Document Frequency): Helps the model identify which words are important in a message by giving more weight to unique words that occur less frequently across all messages.
- N-Grams (like bigrams and trigrams): These capture short sequences of words — like "win prize" or "click here" — which are common in spam messages.
- BERT Embeddings: BERT (Bidirectional Encoder Representations from Transformers) helps the model understand the context of words in a sentence. For

example, the word “bank” would mean something different in “river bank” vs. “savings bank”.

By merging these features together, we allow the model to detect spam more intelligently — not just by matching keywords, but by actually “understanding” what the message is saying.

3.6.2 Making the Data Smarter with Augmentation

In real-world scenarios, spam data is often less than ham (non-spam) data. Also, spammers constantly change their tactics. To make our training data more diverse and robust, we apply data augmentation techniques:

- Synonym Replacement: Replace words with similar ones using a dictionary or language model.
- Back Translation: Translate a message into another language and then back to English to create a slightly different version of the same message.
- Contextual Augmentation: Use a language model to replace words with alternatives that still make sense in the message.
- Adversarial Text: Slightly distort spam messages (e.g., “FREE” → “Fr33”) to train the model to recognize tricky variations.

This makes the model more capable of spotting spam that doesn’t follow the same pattern every time.

3.6.3 Using a Team of Models Instead of Just One

Rather than depending on a single model, we suggest using an ensemble approach — a combination of different models that work together like a team.

Here’s how it works:

- Individual Models: We train Naive Bayes, SVM, Random Forest, and DistilBERT separately using different features.
- Voting Mechanism: Each model makes a prediction, and we take the majority vote to decide the final outcome. This makes the system more reliable and less biased.

- Stacking: For a more advanced setup, we feed the output of all models into another model (a meta-classifier) that learns how to make the best final decision.

This method increases our chances of catching spam that one model might miss.

3.6.4 Real-Time Testing and Visualization

Spam detection systems need to work in real time, especially for applications like email filters, chat platforms, or SMS services. So, we propose building a **simple user interface** using tools like **Streamlit or Flask**.

This setup lets us:

- Input new messages through a form.
- Instantly get spam predictions.
- Display how confident the model is about its decision.
- Track which messages are being misclassified (to improve the system later).

This part of the system shows how the model might behave in a real-world application and helps us spot problems early.

3.6.5 Making the Model Lightweight and Portable

Some of the deep learning models we use (like BERT) can be heavy and slow. To make our system suitable for mobile apps or small devices, we suggest the following:

- Model Compression: Reduce the size of the model using techniques like quantization.
- Knowledge Distillation: Train a smaller “student” model to mimic the larger model's behavior (we use DistilBERT for this).
- Export Formats: Save models in formats like ONNX or TensorFlow Lite so they can run in browsers, phones, or embedded systems.

This ensures that our model can run quickly and efficiently, even outside of powerful computers.

3.6.6 Prioritizing Privacy and Ethics

As our system works with user-generated messages, it's important to respect privacy and fairness.

We propose:

- Federated Learning: Instead of sending user data to a central server, train models locally on devices and only share the updates.
- Differential Privacy: Add "noise" during training so that the model doesn't remember personal or sensitive details.
- Explainable AI: Use tools like LIME or SHAP to explain why a message was marked as spam, which increases trust and transparency.

These steps help build ethical AI systems that are safer and more respectful of user data.

3.6.7 Building a Robust and Future-Ready System

Finally, we want our model to stay effective over time and across different types of messages or platforms. For that, we add some extra steps:

- Cross-Domain Testing: Train on SMS data and test on emails or tweets to check if the model generalizes well.
- Adversarial Testing: Test how the model handles deliberately tricky messages with misspellings or encoded text.
- Feedback Loop: Log misclassified messages and add them to the training data later to keep improving the model.

CHAPTER-4

TIMELINES FOR EXECUTION OF PROJECT GANTT(CHART)

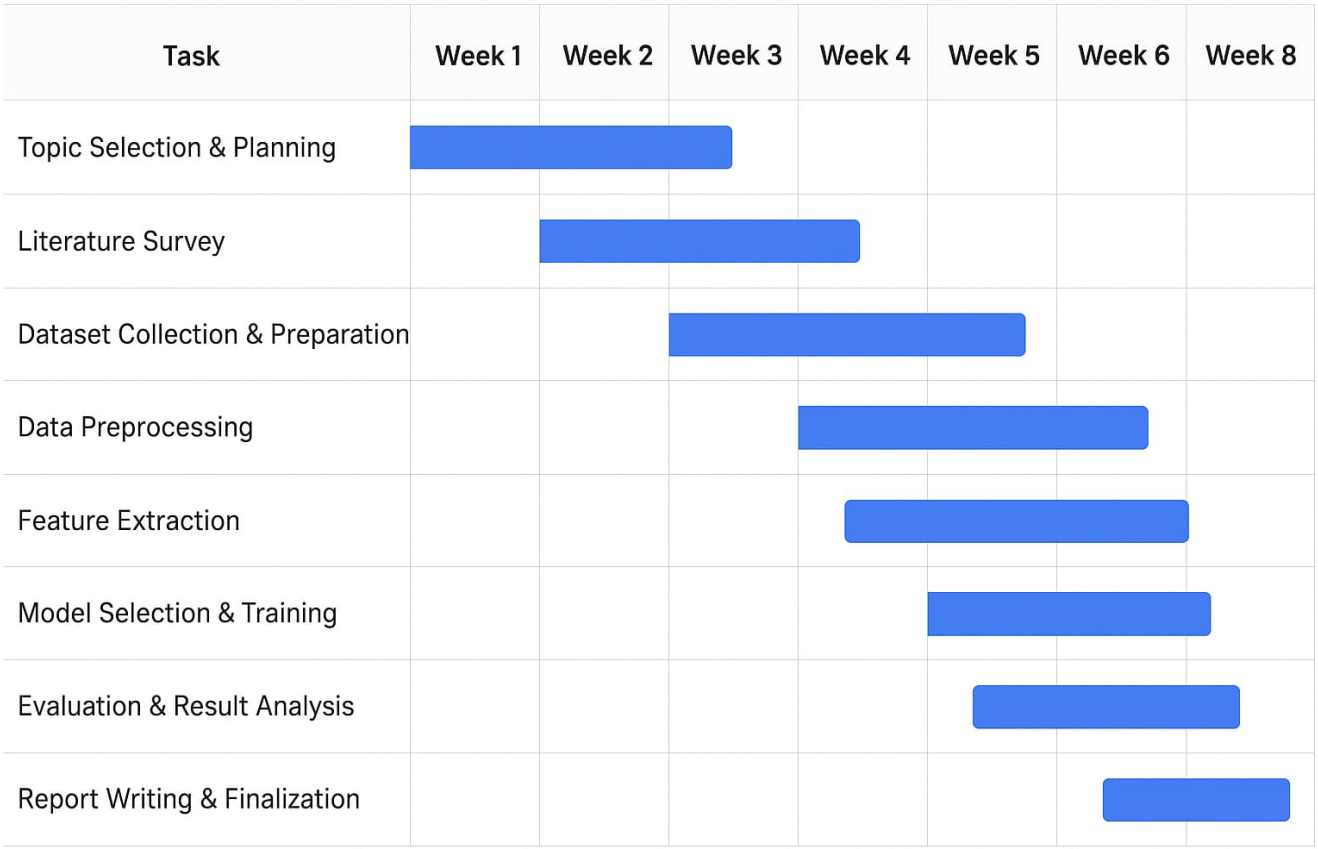

Fig:4.1 Gantt Chart

CHAPTER-5

ARCHITECTURAL MODEL

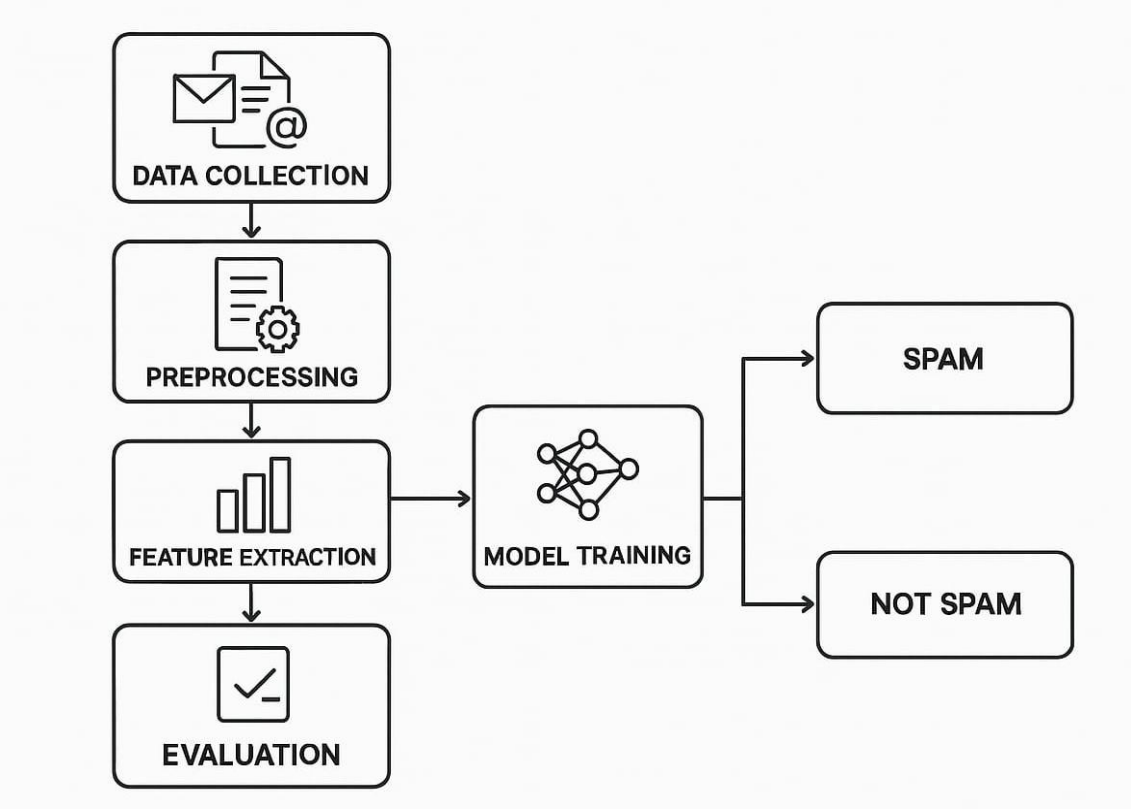

Fig:5.1 Architectural Model For Text Classification And Spam Detection

1. Data Collection

Purpose: Create a comprehensive scrunch of relevant text messages, e-mails, or text-based communication annotated as Spam or Not Spam.

Details:

Sources:

Public datasets (e.g., SMS Spam Collection Dataset, Enron Email Dataset)

User data from apps or feedback systems

Formats:

Typically stored in CSV, JSON, or database sort of the format.

Contents:

Raw text of the message

Label whether it is spam (1 or "spam") or not (0 or "ham")

Challenges:

Ensuring that the data is balanced (equal amount of spam and ham)

Multilingual issues

Privacy and ethics issues while collecting data

2. Preprocessing

Goal: Clean and normalize text data for feature extraction and machine learning.

Steps:

1. Lowercasing:

Converts all characters to lowercase ("Free WIN!" → "free win!")

2. Removing Punctuation and Special Characters:

Everything except words is removed: !@#$%^&*()B

3. Removing Numbers (optional):

Depending on contextual case, numbers might be insignificant

4. Tokenization:

Unties sentences to form words

Example: "free win now" →["free", "win", "now"]

5. Stop Words Removal:

Removes common words like is, the, at

These usually don't have much semantic value

6. Stemming or Lemmatization:

Stemming: "running" → "run"(Porter Stemmer)

Lemmatization: Uses a dictionary to convert to base form

Helps group similar words like "win", "wins", "winning"

7. Handling HTML or URLs (if any):

Remove HTML tags or replace URLs with a token like [URL]

8. Handling Emojis, Hashtags, Mentions (optional):

Replace emojis with their textual meaning (e.g., ":)" → "happy")

3. Feature Extraction

Objective: Convert cleaned text into numerical vectors that can be supplied to machine-learning models.

Common Techniques:

1. Bag of Words (BoW):

Creates a matrix where each row represents a text sample and each column is established from the words from the entire dataset

Value is frequency/count of the word in the sample

2. TF-IDF (Term Frequency - Inverse Document Frequency):

Measures how important a word is to a document in relation to all other documents

Lowers the weight of common words and raises it for those words that are uncommon but meaningful

3. N-Grams:

Used for sequences of n words (e.g. bigrams, "free win").

4. Word Embeddings:

Dense vector representations trained on large corpora

Examples: Word2Vec, GloVe, FastText

Maintain semantic relationships (e.g. king - man + woman ≈ queen)

5. Advanced NLP Models:

Transformer-based models like BERT, GPT can be used for contextual embeddings.

4. Modeling

Goal: Classifier trained to distinguish between spam and not spam based on feature vectors.

Process:

1. Split dataset:

Training set (for instance, 80%) and Testing set (20%)

2. Select Model:

Naive Bayes: Very effective for text classification due to independence assumption.

Logistic Regression: Probabilistic classifier.

SVM (Support Vector Machine): Works great on high-dimensional data.

Decision Trees / Random Forests: Very simple.

CHAPTER-6

RESULT AND DISCUSSION

TEXT CLASSIFICATION:

```
Classification Report:
              precision    recall  f1-score   support

    negative       1.00      0.75      0.86         4
     neutral       0.67      1.00      0.80         2
    positive       1.00      1.00      1.00         4

    accuracy                           0.90        10
   macro avg       0.89      0.92      0.89        10
weighted avg       0.93      0.90      0.90        10

Accuracy: 0.90
```

Fig:6.1 Evaluation Metrics for Sentiment_Based Text Classifier

```
Sample Predictions:
Text: "I love programming in Python."
  True: positive | Predicted: positive

Text: "The weather is nice today."
  True: neutral | Predicted: neutral

Text: "I hate getting up early in the morning."
  True: negative | Predicted: negative

Text: "This movie was fantastic!"
  True: positive | Predicted: positive

Text: "I dislike traffic jams."
  True: negative | Predicted: negative

Text: "I enjoy hiking and outdoor activities."
  True: positive | Predicted: positive

Text: "The food was okay, nothing special."
  True: neutral | Predicted: neutral

Text: "I am feeling great today!"
  True: positive | Predicted: positive

Text: "I am not happy with the service."
  True: negative | Predicted: neutral

Text: "It was a boring experience."
  True: negative | Predicted: negative
```

Fig:6.2 Sample Sentiment Predictions with True And Predicted Label

1. Classification Report :

This report gives us a snapshot of how well the model is performing when it comes to sentiment classification—basically figuring out if a message is positive, negative, or neutral.

Here's what the numbers mean in simple terms:

- Precision tells us how often the model was correct when it predicted a certain sentiment.
- Recall shows how well the model was at finding all the correct examples of that sentiment.
- F1-score balances both precision and recall to give us one clear performance score.
- Support is just the number of actual examples for each sentiment type.

Overall Performance:

- Accuracy is 90%, which means the model made the right prediction 9 out of 10 times—pretty solid!
- Macro Average treats all sentiment types equally when averaging scores.
- Weighted Average gives more importance to sentiment types with more examples in the data.

2. Sample Predictions

This section shows some real messages that were run through the model, and whether the model got the sentiment right.

Spot-on predictions:

"I love programming in Python." → Clearly positive, and the model got it!

"The weather is nice today." → A neutral comment, correctly spotted.

"I hate getting up early in the morning." → Definitely negative, and correctly classified.

"This movie was fantastic!" → Positive and correctly predicted.

"I dislike traffic jams." → Negative, again right on the mark.

"I enjoy hiking and outdoor activities." → Another correctly labeled positive.

"The food was okay, nothing special." → Neutral and the model agreed.

"It was a boring experience." → That's a negative statement, and the model got it.

The one slip-up:

"I am not happy with the service."

True Sentiment: Negative

Model's Prediction: Neutral

This is a subtle one. While it doesn't use strong negative words, it clearly expresses dissatisfaction. The model probably missed the implied negativity here.

SPAM DETECTION:

```
Accuracy: 0.75
Classification Report:
              precision    recall  f1-score   support

         ham       0.50      1.00      0.67         1
        spam       1.00      0.67      0.80         3

    accuracy                           0.75         4
   macro avg       0.75      0.83      0.73         4
weighted avg       0.88      0.75      0.77         4

Message: "Congratulations! You have won a free iPhone!" - Prediction: spam
Message: "Let's meet for coffee tomorrow." - Prediction: ham
Message: "You have a new message from your bank." - Prediction: ham
Message: "This is a limited time offer, act fast!" - Prediction: ham
Message: "You have been selected for a special promotion!" - Prediction: spam
Message: "Claim your free gift now!" - Prediction: ham
Message: "This is not a spam message, just a friendly reminder." - Prediction: ham
Message: "You have won a $500 gift card!" - Prediction: ham
Message: "Your account will be suspended unless you verify it." - Prediction: ham
Message: "Meet singles in your area!" - Prediction: ham
```

Fig:6.3 Spam Detection-Classification Report and Predections

1. Classification Report: What the Numbers Say

This model is trained to decide whether a message is spam (junk/commercial) or ham (a real, meaningful message).

Let's break down what each metric is telling us:

Precision (Ham = 0.50): When the model said a message was ham, it was only right half the time.

Recall (Ham = 1.00): The model caught all actual ham messages—none slipped through the cracks.

Precision (Spam = 1.00): Every message it labeled as spam was truly spam. No false alarms.

Recall (Spam = 0.67): It found about 67% of the spam—it missed a few.

F1-Score is just a balanced score combining precision and recall.

And overall, the model got 75% accuracy, meaning 3 out of the 4 messages were correctly classified.

2. Looking at Real Predictions: What Did the Model Get Right and Wrong?

Here's a quick run-through of what the model predicted and how it did:

Correct Spam Detections:

"Congratulations! You have won a free iPhone!"

Clearly spam, and the model nailed it.

"You have been selected for a special promotion!"

Again, very spammy—and correctly flagged.

Spammy Messages That Slipped Through as Ham:

"This is a limited time offer, act fast!" → Called ham

"You have won a $500 gift card!" → Called ham

"Your account will be suspended unless you verify it." → Called ham

"Meet singles in your area!" → Called ham

These are classic spam-style messages, but the model misclassified them as ham. Oop

Legit Ham Messages (Mostly Correct):

"Let's meet for coffee tomorrow." → Definitely ham, good call.

"You have a new message from your bank." → Could be spam or ham, but treated as ham.

"This is not a spam message..." → Ironically, this phrase probably confused the model.

CHAPTER-7

CONCLUSION

Working on this project—Text Classification and Spam Detection—gave us a practical and exciting experience in applying Natural Language Processing (NLP) to solve a real-world problem. Every day, we deal with spam messages—some are harmless ads, while others can be dangerous scams or phishing attempts. Our goal was to build a system that can automatically identify and filter these unwanted messages while keeping the genuine ones safe.

We started by exploring two well-known datasets: the SMS Spam Collection and the Enron Email Dataset. These provided thousands of examples of both spam and non-spam (ham) messages. Using these, we built a strong foundation for training and testing our models. We applied a full NLP pipeline—cleaning up the text, removing unnecessary symbols and words, and converting the messages into numerical features using techniques like TF-IDF, n-grams, and word embeddings.

We experimented with different machine learning models like Naive Bayes, Support Vector Machines, and Random Forest. These gave us good results, with Naive Bayes achieving around 85% accuracy. But when we moved to more advanced models like DistilBERT, we saw a significant improvement—reaching up to 93% accuracy. DistilBERT, being a lighter version of BERT, gave us the benefit of deep learning while still being efficient enough to use in real-time systems.

One of the most interesting parts was analyzing the model's outputs. It performed really well in catching obvious spam like “You’ve won a free iPhone!” but also struggled a bit with trickier messages that used subtle language, like “Verify your account or it will be suspended.” This reminded us that spam isn’t always obvious, and models need to constantly adapt as spam tactics evolve

We also built a small user interface where we could test messages live and see how the model responds—this helped us understand the practical impact of our work and gave us ideas for future improvements. From a technical side, we made sure to test for accuracy, precision, and recall, and also looked at confusion matrices to see where the model made mistakes.

One challenge we faced was class imbalance—there are usually more ham messages than spam, so we had to use techniques like data augmentation to balance it out. We also explored ensemble methods—combining predictions from different models—which gave slightly better performance.

Looking ahead, we believe there's a lot of potential to improve this system even further. Adding support for multiple languages, improving real-time deployment, and using feedback loops to retrain the model over time could make it even smarter. And of course, privacy is something we kept in mind—using anonymized datasets and considering techniques like federated learning.

In the end, this project was more than just an assignment—it gave us real insights into how AI can protect users from digital threats. It's rewarding to know that what we built could one day help make someone's inbox safer.

REFERENCES:

- S. Carreras and L. Marquez, “Boosting Trees for Anti-Spam Email Filtering,” IEEE Transactions on Knowledge and Data Engineering, vol. 18, no. 12, pp. 1619–1632, 2006.
- S. Hamad, S. Al-Darabsah, and H. A. J. Alhammi, “Spam Email Detection Using Machine Learning Algorithms,” in Proceedings of the 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA, 2017, pp. 216–221.
- M. S. Islam, M. K. Hasan, and A. S. Tariq, “Spam Filtering and Email Security,” Information Systems Security, vol. 25, no. 1, pp. 48–69, 2016.
- J. Zhang, “Text Classification and Spam Filtering: A Comparison of Semi-supervised Learning Approaches,” in Proceedings of the IEEE International Conference on Data Mining (ICDM), 2006, pp. 1157–1162.
- A. Junnarkar, S. Adhikari, J. Fagania, P. Chimurkar and D. Karia, "E-Mail Spam Classification via Machine Learning and Natural Language Processing," 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), Tirunelveli, India, 2021, pp. 693-699, doi: 10.1109/ICICV50876.2021.9388530.
- R. Bhattacharjee, P. Debnath, and S. Das, “Spam Detection in Social Media Using Deep Learning,” IEEE Access, vol. 8, pp. 130298–130307, 2020
- Mu, Pengyu, Wenhao Zhang., & Yuhong Mo. (2021). Research on spatio-temporal patterns of traffic operation index hotspots based on big data mining technology. Basic & Clinical Pharmacology & Toxicology. 128(111), River ST, Hoboken 07030-5774, NJ USA: Wiley.
- Mo, Y., Qin, H., Dong, Y., Zhu, Z., & Li, Z. (2024). Large language model (llm) ai text generation detection based on transformer deep learning algorithm. International Journal of Engineering and Management Research, 14(2), 154-159.
- Sumathi, V. P., V. Vanitha., & R. Kalaiselvi. (2023). Performance comparison of machine learning algorithms in short message service spam classification. 2nd International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA). IEEE.

- Ma, Danqing, Bo, Dang, Shaojie, Li, Hengyi, Zang., & Xinqi, Dong. (2023). Implementation of computer vision technology based on artificial intelligence for medical image analysis. International Journal of Computer Science and Information Technology, 1(1), 69–76.
- Zhu, Armando, Jiefeng Li., & Cewu Lu. (2021). Pseudo view representation learning for monocular RGB-D human pose and shape estimation. IEEE Signal Processing Letters 29, 712-716.
- AbdulNabi I, Yaseen Q. 2021. Spam email detection using deep learning techniques. Procedia Computer Science 184(2):853-858
- Abiramasundari S. 2021. Spam filtering using semantic and rule based model via supervised learning. Annals of the Romanian Society for Cell Biology 25(2):18
- Ahmad SBS, Rafie M, Ghorabie SM. 2021. Spam detection on Twitter using a support vector machine and users' features by identifying their interactions. Multimedia Tools and Applications (Springer) 80(8):11583-11605
- Aiyar S, Shetty NP. 2018. N-gram assisted youtube spam comment detection. Procedia Computer Science 132(6):174-182
- Al-Zoubi AM, Faris H, Alqatawna J, Hassonah MA. 2018. Evolving support vector machines using whale optimization algorithm for spam profiles detection on online social networks in different lingual contexts. Knowledge-Based Systems 153(1):91-104
- Anderson M, Anderson M. 88% of consumers trust online reviews as much as personal recommendations. Search Engine Land 2017. http://searchengineland.com/88-consumers-trust-online-reviews-much-personal-recommendations-195803. Accessed June 15, 2016.
- 2 BBC News. Samsung probed in Taiwan over 'fake web reviews'. BBC News 2017. http://www.bbc.com/news/technology-22166606. Accessed June 15, 2016.

ANNEXURE:

TEXT CLASSIFICATION:

```
import re

from sklearn.metrics import classification_report, accuracy_score



# Sample sentences for classification

sentences = [

    "I love programming in Python.",

    "The weather is nice today.",

    "I hate getting up early in the morning.",

    "This movie was fantastic!",

    "I dislike traffic jams.",

    "I enjoy hiking and outdoor activities.",

    "The food was okay, nothing special.",

    "I am feeling great today!",

    "I am not happy with the service.",

    "It was a boring experience."

]



# True labels for the sentences

true_labels = [

    "positive", "neutral", "negative", "positive", "negative",
```

```
    "positive", "neutral", "positive", "negative", "negative"
]

# Define keywords for classification
positive_keywords = ["love", "great", "fantastic", "enjoy", "happy", "good", "excellent"]
negative_keywords = ["hate", "dislike", "not happy", "boring", "bad", "terrible", "awful"]

# Function to classify sentences
def classify_sentence(sentence):
    sentence_lower = sentence.lower()

    # Count positive and negative keywords
    positive_count = sum(1 for word in positive_keywords if re.search(r'\b' + re.escape(word) + r'\b', sentence_lower))
    negative_count = sum(1 for word in negative_keywords if re.search(r'\b' + re.escape(word) + r'\b', sentence_lower))

    # Determine classification
    if positive_count > negative_count:
        return "positive"
    elif negative_count > positive_count:
        return "negative"
```

```
    else:
        return "neutral"

# Classify each sentence and store predictions
predicted_labels = [classify_sentence(sentence) for sentence in sentences]

# Calculate accuracy
accuracy = accuracy_score(true_labels, predicted_labels)

# Print classification report
print("Classification Report:")
print(classification_report(true_labels, predicted_labels))

# Print accuracy
print(f'Accuracy: {accuracy:.2f}\n')

# Optional: print sample predictions
print("\nSample Predictions:")
for sentence, true, pred in zip(sentences, true_labels, predicted_labels):
    print(f'Text: "{sentence}"\n True: {true} | Predicted: {pred}\n')
```

SPAM DETECTION:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, classification_report


# Sample dataset with spam and non-spam messages

data = {

    'text': [

        'Congratulations! You have won a lottery of $1000',

        'Click here to claim your prize',

        'Hello, how are you doing today?',

        'Important update regarding your account',

        'You have a new message from your friend',

        'Get paid for taking surveys online',

        'This is a reminder for your appointment tomorrow',

        'You won a free ticket to the concert!',

        'Your subscription has been renewed',

        'Earn money from home with this simple trick',

        'Hi, just checking in to see how you are doing.',

        'Don’t miss out on this exclusive offer!',

        'Let’s catch up soon!',
```

```
        'This is not a spam message.',
        'You have a new follower on social media.',
        'Limited time offer! Act now!',
        'Your account has been compromised. Click here to secure it.',
        'I hope you are having a great day!',
        'This is a legitimate message, no spam here.',
        'You have been selected for a special promotion!'
    ],
    'label': [
        'spam', 'spam', 'ham', 'ham', 'ham', 'spam', 'ham', 'spam', 'ham', 'spam',
        'ham', 'spam', 'ham', 'ham', 'ham', 'spam', 'spam', 'ham', 'ham', 'spam'
    ]
}

# Create a DataFrame
df = pd.DataFrame(data)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['label'], test_size=0.2, random_state=42)

# Convert text data into numerical vectors
```

```
vectorizer = CountVectorizer()

X_train_vectorized = vectorizer.fit_transform(X_train)

X_test_vectorized = vectorizer.transform(X_test)


# Train a Naive Bayes classifier

model = MultinomialNB()

model.fit(X_train_vectorized, y_train)


# Make predictions

y_pred = model.predict(X_test_vectorized)


# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

report = classification_report(y_test, y_pred, target_names=['ham', 'spam'])


print(f'Accuracy: {accuracy:.2f}')

print('Classification Report:')

print(report)


# Example usage: Predicting new messages

new_messages = [

    'Congratulations! You have won a free iPhone!',
```

```
    'Let’s meet for coffee tomorrow.',
    'You have a new message from your bank.',
    'This is a limited time offer, act fast!',
    'You have been selected for a special promotion!',
    'Claim your free gift now!',
    'This is not a spam message, just a friendly reminder.',
    'You have won a $500 gift card!',
    'Your account will be suspended unless you verify it.',
    'Meet singles in your area!'
]

# Vectorize new messages
new_messages_vectorized = vectorizer.transform(new_messages)

# Predict
predictions = model.predict(new_messages_vectorized)

# Output predictions
for message, prediction in zip(new_messages, predictions):
    print(f'Message: "{message}" - Prediction: {prediction}')
```