Email Spam Detection using BERT

Tapaswi Satyapanthi
College of Engineering
Northeastern University
satyapanthi.t@northeastern.edu

1 Introduction

This project focuses on detecting spam emails using a transformer-based model, specifically BERT (Bidirectional Encoder Representations from Transformers). The objective is to build a robust classifier that can accurately differentiate between spam and non-spam emails. The tools used for model development include `Pandas`, `PyTorch`, `TensorFlow`, `Transformers`, and `Scikit-Learn`, while model deployment is facilitated using `Huggingface Spaces`, `Docker`, and `FastAPI`. For training and development, Google Colab was used as the primary platform.

2 Why BERT?

BERT, based on transformer architecture, excels in understanding the context in textual information, which is crucial for spam detection. Unlike traditional models, BERT processes the entire text bidirectionally, allowing it to capture nuanced meanings and relationships between words. This contextual understanding significantly improves performance in natural language processing (NLP) tasks such as spam detection, where subtle differences in wording can drastically change the meaning of an email.

3 Data Set and Processing

The **Enron-Spam** dataset was used for training and testing the model. This dataset consists of thousands of emails from the Enron Corporation, categorized as either spam or ham (non-spam). The raw data was stored in `.txt` files, so I extracted the text from these files and saved them into a `.csv` format using `Pandas`. Figure 1 shows the sample of extracted Data Set

3.1 Class Imbalance

The dataset initially had a class imbalance with 4500 spam emails and only 1500 ham emails. To mitigate this issue, I downsampled the spam emails to 1500 to create a balanced dataset. For BERT, a balanced dataset

	content	is_spam
3086	Subject: application approval for kernel @ vge...	1
2820	Subject: ets : the next generation\nclick here...	0
3480	Subject: congratulations\nfrom : the desk of t...	1
4261	Subject: dynegydirect maintenance update\ndyne...	0
4180	Subject: winning notification\nworldwinnings h...	1
1251	Subject: mail : from west africa relief agency...	1
5055	Subject: good day\ndear sir / madam ,\nmy name...	1
1426	Subject: time may be running out\nhello ,\nwe ...	1
2599	Subject: 75 % off for all new software .\npopu...	1
59	Subject: our hot picks triple on excelient bou...	1

Figure 1: Email Spam or Ham DataFrame

of 1500 records per class is sufficient because it has been pre-trained on a large corpus and excels in context understanding. Figures 2 and 3 show the class distribution before and after downsampling.

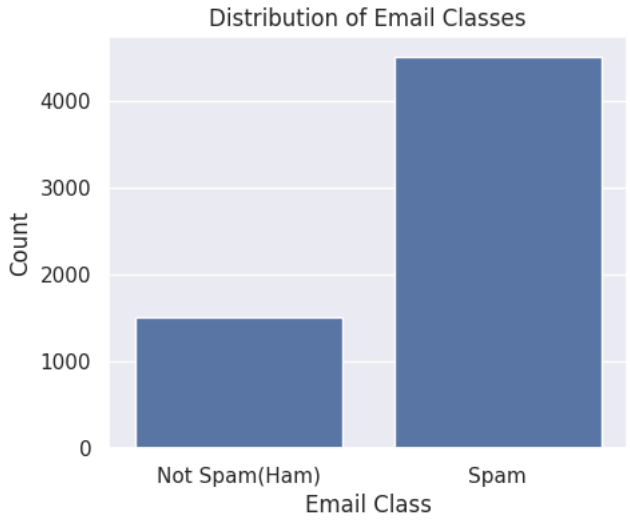

Figure 2: Class Distribution Before Downsampling

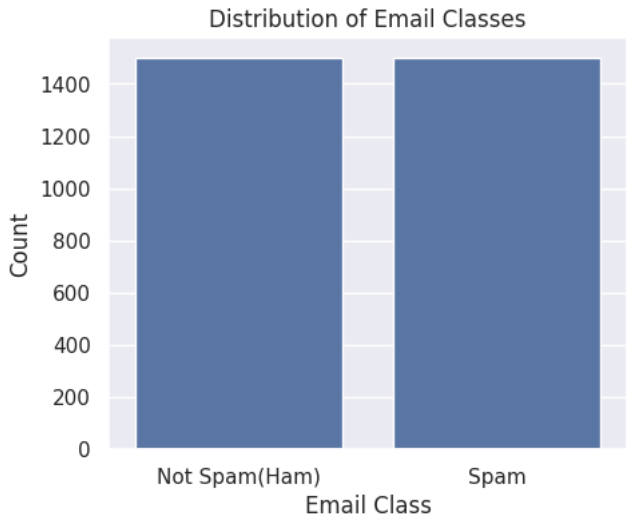

Figure 3: Class Distribution After Downsampling

3.2 Text Processing

Text preprocessing was done using regular expressions to remove URLs, HTML tags, non-alphanumeric characters, and extra spaces. Unlike traditional NLP approaches, lemmatization and stopword removal were not necessary because transformer-based models like BERT are capable of capturing context, and removing words could degrade performance.

4 Training and Testing Data Split

The data was split into training and testing sets using an 80:20 ratio. This ensures that the model has sufficient data for training while reserving enough data for evaluating its performance.

5 Fine-tuning the Model

For fine-tuning, I used the `Huggingface Transformers` library and specifically employed the BERT-base-uncased model. BERT-base contains 12 transformer layers and 110 million parameters, which makes it highly

capable of understanding contextual relationships in text. The fine-tuning process involved the following steps:

1. Downloading the pre-trained BERT-base model from Huggingface.
2. Preparing the dataset and tokenizing it using BERT's tokenizer.
3. Fine-tuning the model on the dataset for 3 epochs, which was sufficient to achieve high accuracy.

The model was trained on a CPU, which took about 1.5 hours. On a GPU, the training time was reduced to approximately 3 minutes. A tokenizer and model are both necessary: the tokenizer converts text into tokens that the model can understand, while the model itself handles the context understanding.

6 Model Evaluation

The model was evaluated on the testing dataset and achieved an accuracy of 98%. Figure 4 shows the confusion matrix, and Figure 5 displays the classification report.

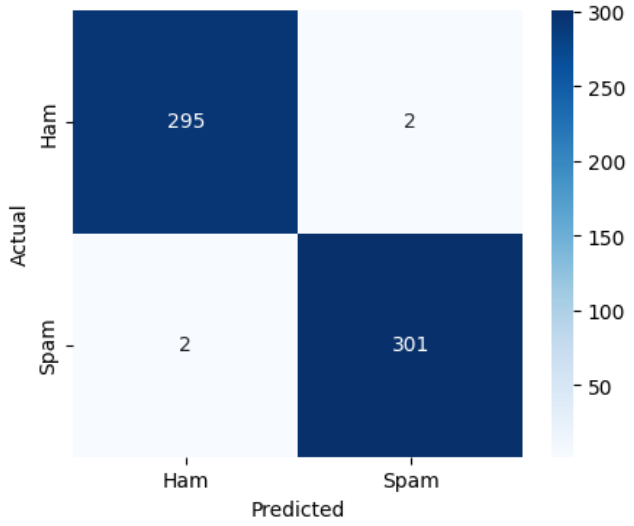

Figure 4: Confusion Matrix

	precision	recall	f1-score	support
0	0.99	0.99	0.99	297
1	0.99	0.99	0.99	303
accuracy			0.99	600
macro avg	0.99	0.99	0.99	600
weighted avg	0.99	0.99	0.99	600

Figure 5: Classification Report

7 Model Saving

The final model was saved in the format that the `Transformers` library understands, which includes saving the model weights and the tokenizer separately. This modular saving format allows for easier loading and deployment of the model.

8 Model's Memory Usage

This BERT model contains over 100 million parameters (109,483,778). The memory usage is calculated as follows:

- Memory in bytes = **Total no. of parameters × 4 bytes = 109483778 × 4 = 437,935,112 bytes**
- Memory Consumption in MB = **(437,935,112 / 1024^2) = 417.65 MB**
- Memory Consumption in GB = **(417.65 / 1024) = 0.408 GB**

9 Model Deployment

The model was deployed using `Huggingface Spaces`, which offers a streamlined platform for hosting machine learning models. Huggingface Spaces integrates with Git repositories and provides Docker support, enabling a containerized deployment for scalability and environment consistency.

9.1 Step-by-step Deployment Process

The following steps outline the deployment process:

9.1.1 Step 1: Dockerization of the Model

Docker was used to create a containerized environment for the BERT model. A Docker container ensures that all necessary dependencies, such as the Huggingface Transformers library, are available in the deployment environment. The Docker image contains the environment setup along with the trained model.

9.1.2 Step 2: API Creation using FastAPI

To serve the model for inference, a REST API was developed using `FastAPI`. FastAPI is a lightweight, high-performance web framework that allows users to send email text via POST requests. It loads the pre-trained model, runs inference, and returns whether the email is spam or not. This API setup is essential for easy integration with other services or applications.

9.1.3 Step 3: Hosting on Huggingface Spaces

After Dockerizing the model and setting up the API, the next step was to push the project to a Git repository linked with `Huggingface Spaces`. Huggingface Spaces detects the Dockerfile and automatically builds and hosts the model in a scalable environment. It provides an API endpoint that can be accessed by external users or applications for inference.

- **Hugging Face Space App**: https://huggingface.co/spaces/Tapaswi24/bert-spam-classification
- **Hugging Face Space Git Repo**: https://huggingface.co/spaces/Tapaswi24/bert-spam-classification/tree/main

10 Inference

After deployment, the API can be accessed at the URL provided by Huggingface Spaces. Users can send POST requests with email text, and the API will respond with whether the email is spam or not.

10.1 API Endpoint

The deployed REST API provides an endpoint for spam detection. The endpoint is a POST request to `/is-spam` that accepts a JSON body with the email text.

```
POST https://Tapaswi24-bert-spam-classification.hf.space/is-spam
Request Body (JSON): { "text": "Your email content here" }
```

An example curl request:

```
curl -X POST https://Tapaswi24-bert-spam-classification.hf.space/is-spam \
-H 'Content-Type: application/json' \
-d '{"text": "Congratulations! You have won a free iPhone!"}'
```

The API returns the following response:

```
{"spam": true}
```

11 Conclusion

This project demonstrates how BERT can be effectively fine-tuned for email spam detection. The model was deployed using FastAPI and Docker on Huggingface Spaces, showcasing an end-to-end deployment process that ensures scalability and ease of use.