CMPSC 497 - Midterm Project

Spam Detection using Deep Learning Techniques - Shreya Buddharaju and Yuv Boghani

Abstract

This project explores the development of a system for detecting spam messages using deep learning methods. The system aims to classify SMS messages as either "spam" or "ham" (non-spam) by leveraging word embeddings and a Bidirectional Long Short-Term Memory architecture. This report details the project's methodology, results, and key learnings.

Problem Definition

The increasing prevalence of spam messages poses a significant issue for communication systems. These unwanted messages can be disruptive, pose security risks, and contribute to the spread of misinformation. This project addresses this issue by developing a system capable of automatically identifying spam messages.

Dataset Definition

The project utilizes the "SMS Spam Collection" dataset on Kaggle, which consists of a collection of SMS messages labeled as either "spam" or "ham." The dataset provides a real-world representation of the types of messages encountered in spam filtering scenarios.

Data Preprocessing Steps

The following steps were taken to prepare the data for training:

- *Data Loading and Cleaning*: The dataset was loaded using the pandas library, and the columns were renamed for clarity.

- *Tokenization*: The text messages were tokenized using the Keras Tokenizer, with a vocabulary size of 10,000 words. An out-of-vocabulary token was used to handle unknown words.

- *Sequence Padding*: The tokenized sequences were padded to a uniform length of 50 tokens to ensure compatibility with the neural network.

- *Label Encoding*: The categorical labels ("spam" and "ham") were converted into numerical values (0 and 1).

This dataset was well maintained, easily accessible and fit our use case (variety in messages segregated into two labels for spam and non-spam) perfectly. We also considered various other datasets (such as the UCI SMS Spam dataset and the Enron Spam Dataset) but no freely available datasets online had more unique values for our training and were implemented as simply as the SMS Spam Collection file from Kaggle.

Methodology

- **Word Embeddings (GloVe):**
  Word embeddings provide a way to represent words as numerical vectors, capturing semantic relationships between words. This project employs pre-trained GloVe (Global Vectors for Word Representation) embeddings to initialize the word representations. The "glove-wiki-gigaword-100" model was utilized using the gensim.
  An embedding matrix was created, mapping each word in the vocabulary to its corresponding GloVe vector. Out-of-vocabulary words were assigned a zero vector. The embedding layer was configured to be non-trainable, leveraging the knowledge captured in the pre-trained embeddings.
- **Neural Network Architecture:**
  A Bidirectional Long Short-Term Memory (Bi-LSTM) network was chosen for its ability to model sequential data and capture long-range dependencies in text. The model architecture is as follows:
  1. *Embedding Layer*: Converts word tokens into GloVe vectors.
  2. *Bi-LSTM Layer (128 units)*: Processes the input sequence in both directions.
  3. *Bi-LSTM Layer (64 units)*: A second Bi-LSTM layer for further feature extraction.
  4. *Dense Layer (64 units, ReLU activation)*: Introduces non-linearity into the model.
  5. *Output Layer (1 unit, Sigmoid activation)*: Produces a probability score indicating the likelihood of a message being spam.
- **Training Procedure**:
  The model was trained using the following settings:
  - Optimizer: Adam optimizer
  - Loss Function: Binary cross-entropy loss
  - Metrics: 96% Accuracy
  - Batch Size: 64
  - Epochs: 5 (due to time and resource constraints)

- Validation Split: 10% of the training data

Results and Evaluation

- **Evaluation Metrics**:
  The model's performance was evaluated using the following metrics:
  - *Accuracy*: The percentage of correctly classified messages.
  - *Precision*: The ability of the model to correctly identify spam messages.
  - *Recall*: The ability of the model to identify all actual spam messages.
  - *F1-Score*: A balanced measure of precision and recall.
- **Performance Summary**:
  The model achieved the following results on the test set:
  - *Validation Accuracy*: 94%
  - *Precision*: 0.98 for spam and 0.92 for ham
  - *Recall*: 0.99 for spam and 0.90 for ham
  - *F1-Score*: 0.99 for spam and 0.91 for ham
- **Model Strengths and Limitations**:
  The model demonstrates the ability to effectively classify spam messages. However, potential limitations include sensitivity to hyperparameter selection and the possibility of overfitting.

Analysis and Experimentation

- **Hyperparameter Exploration**:
  Experiments were conducted to investigate the impact of different hyperparameters on model performance, including:
  - *Number of LSTM Units*: Variations in the number of LSTM units in the Bi-LSTM layers were tested (e.g., 64, 128, 256). We observed that increasing the number of units initially improved performance, but beyond 128 units there was no significant improvement so we left it at 128.
  - *Learning Rate*: Different learning rates were explored using the Adam optimizer (e.g., 0.01, 0.001, 0.0001). A learning rate of 0.001 was found to provide a good balance as that gave us fast training times and the best accuracy as compared to all the different variations.
- **Model Architecture Variations**:
  Several model architectures were explored to compare their performance and effectiveness for the spam detection task:
  - *Initial LSTM Experiments*: Early experiments involved using a single-layer LSTM network. While this model showed promise, it struggled to capture the complex relationships between words and phrases, resulting in lower

accuracy compared to the Bi-LSTM model. After doing some more reading and research online we came to the conclusion that it would be ideal to use a Bi-LSTM model instead.

- *CNN Attempts and Setbacks*: Attempts were made to incorporate Convolutional Neural Networks (CNNs) for feature extraction before feeding the data into the LSTM layers. However, initial CNN configurations resulted in significantly decreased performance and increased training time. We also had a lot of dimension errors which we couldn't figure out due to which we abandoned this in the end.
- *Hybrid Approach with Bi-LSTMs* : We decided to utilize Bidirectional LSTMs for the best results. By processing the input sequence in both forward and backward directions, the Bi-LSTM captured contextual information more effectively. This architecture yielded the best performance in terms of accuracy, precision, recall, and F1-score, and that is what the core architecture that is implemented in the final iteration.

Lessons and Experience

We learned a lot from this experience as we initially decided on this task and then wanted to try Named Entity Recognition. However, we struggled a lot with that and then came back to this. We had to fix a lot of problems on the initial code that we had started on and we struggled a lot with padding,masking and the shape of the inputs for the model which initially led to a lot of errors and terrible accuracy scores. We also had a lot of problems with mapping the train and test data due to a very small oversight and had not done it categorically to the labels present. We generally tried to solve a lot of these issues by adding more code to filter and refine the issues however all of this led to more errors and more confusion.

In the end, we managed to make it work by identifying the most crucial portions of our code and pipeline. Isolating them completely and then building from there, this helped us identify a lot of small mistakes we made initially and didn't look back at. We then managed to add more layers and figure out the solution.

Conclusion

This project can be expanded upon in the future as it uses only one dataset for consistency in this project but by opening up its applications to emails and newsletter, etc as well we can easily manage to make the model more accurate across an array of

different sources. This has been an extremely insightful experience and we can’t wait to try to implement more difficult and interesting techniques in the final project!