D214 Data Analytics Graduate Capstone

Machine Learning SPAM Detection Powered by Enron/TREC Public Spam Corpus

Table of Contents

Problem and Hypothesis Statement

Problem:

Spam presents a significant challenge for both businesses and households. It not only consumes valuable storage space and impedes the identification of critical emails, but it also poses a substantial security risk. Spam emails can serve as conduits for malicious threats such as viruses, worms, phishing scams, and ransomware. Therefore, implementing models capable of accurately detecting spam is essential. These models are not only crucial for enhancing security measures but also for ensuring that email server storage is reserved exclusively for important organizational communications.

Hypothesis:

The machine learning model developed using the 2007 TREC Public Spam Corpus and Enron emails dataset can accurately classify new content as SPAM or non-SPAM (HAM) with an accuracy score of 95% or higher. This suggests that the model has learned effective patterns within the dataset, enabling it to differentiate between SPAM and non-SPAM content significantly better than chance.

Summary of Data-Analysis Process

The data-analysis process for the SPAM detection project can be summarized in several key steps:

Data Collection:

- Source: The dataset was obtained from Kaggle.com, consisting of pre-labeled Enron email contents.
- Labeling: Emails were categorized as 'SPAM' (1) and 'HAM' (0), where HAM refers to non-SPAM emails.
- Dataset Size and Balance: The dataset comprised approximately 84,000 entries, with a nearly even split of 47.38% HAM and 52.62% SPAM, indicating minimal bias.

Data Extraction and Preparation:

- Loading Data: Due to GitHub size limitations, the dataset was split into two parts, which were subsequently merged into a single Polars dataset.
- Missing Data Check: The dataset's completeness was affirmed by the creator and verified using the Polars DataFrame **.null_count()** function, confirming no missing data.
- Label Balance Verification: The balance of labeling was checked using the Polars Series **.hist()** function, revealing a balanced distribution of 39,538 HAM and 43,910 SPAM entries.

Text Data Preprocessing:

- Preparing for Machine Learning: The project utilized Logistic Regression, necessitating the conversion of unstructured text into a structured format.
- Vectorization (Embeddings): Email contents were vectorized through a series of preprocessing steps:
    - Text Normalization: Converting all text to lowercase and removing Unicode characters and emojis.
    - Punctuation Removal: Stripping away punctuation from the text.
    - Tokenization: Breaking down the text into smaller parts or tokens.

- Stop Word Removal: Eliminating common words (e.g., 'the', 'is', 'in') that do not contribute to the specific meaning in the context of SPAM detection.

This structured approach ensured the dataset was optimally prepared for effective machine learning model training and SPAM detection analysis.

Outline of Findings

High Accuracy:

The model exhibits a high accuracy of about 98.96% and a prediction accuracy of approximately 98.27%. This indicates a strong capability in distinguishing SPAM(1) from HAM(0). A Logistical Regression Model can be effectively trained for such classification tasks using datasets like the TREC Public Spam Corpus and Enron emails.

AUC Score:

An AUC (Area Under the Curve) Score of 98.2% suggests that the model has a high true positive rate and a low false positive rate. With regard to SPAM detection this is important as it means we have a low mis-classification (false-positive) of emails as SPAM(1) when they are actually HAM(0)

Precision and Recall:

The precision and recall values are both high across SPAM(1) and HAM(0) classifications. This balance is essential for practical applications to ensure that both spam and legitimate emails are accurately identified.

Error Metrics:

The Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) are relatively low, indicating that the model's predictions are generally close to the actual values.

Limitations of Techniques/Tools Used

In the realm of data extraction and preparation, several tools and techniques were employed, each presenting unique limitations that are essential to consider for optimal application.

Tools:

1. **Python**: Although Python is renowned for its user-friendliness, particularly beneficial for those without a programming background, it is not without its drawbacks. The primary limitation of Python lies in its slower run-time performance, a trade-off for its simplicity and accessibility. This aspect can be particularly challenging in scenarios where speed and efficiency are crucial.

2. **Polars**: As a data manipulation library, Polars excels in handling larger datasets more efficiently than Pandas due to its Rust-based development. However, its relative novelty compared to Pandas means a smaller community support base. This can pose challenges in finding resources or assistance for specific issues or advanced functionalities.

3. **Matplotlib**: This widely used visualization library is not dependent on Pandas, despite their integration. However, Matplotlib's extensive customization capabilities come with a steep learning curve. Users often find themselves needing to consult documentation frequently, even for what might seem like basic operations, indicating a barrier to efficient usage.

4. **Seaborn**: Built atop Matplotlib, Seaborn simplifies certain aspects of plotting, such as creating heatmaps. Nonetheless, this high-level interface approach can sometimes limit the degree of customization available in Matplotlib, potentially restricting the user's ability to achieve specific visual representations.

5. **NLTK (Natural Language Toolkit)**: While NLTK is user-friendly for natural language processing, it struggles with larger datasets, exhibiting reduced performance efficiency and slower processing speeds. This can be a significant hindrance in projects involving substantial amounts of text data.

6. **Emoji Library**: This tool is effective in identifying and removing emojis from text, which is valuable during pre-processing. However, its functionality is limited to this specific task, offering little else in terms of broader text processing or analysis capabilities.

7. **Unidecode**: Useful for converting non-ASCII characters during preprocessing, Unidecode, however, operates solely within the ASCII character set. This means that it can strip away meaning when applied to non-English languages, a notable limitation in multilingual projects.

8. **Scikit-Learn TfidfVectorizer**: Employed for converting unstructured text to structured, vectorized embeddings, the TfidfVectorizer faces a critical limitation in its inability to preserve the sequential order of words. This leads to a potential loss of context after vectorization, which can be detrimental in text analysis where order is important.

In summary, while these tools are instrumental in data extraction and preparation, their limitations must be carefully considered to ensure their effective and appropriate use in different contexts.

Techniques:

1. **Model Accuracy with .score() Method**: The **.score()** method provides a quick assessment of the model's accuracy. While its simplicity is great for a basic overview, it lacks the depth required for a comprehensive understanding of the model's predictive capabilities. It does not offer insights into specific areas where the model may fail or excel, such as how it handles false positives or false negatives.

2. **Predictive Accuracy with .predict() and accuracy_score()**: While these functions offer a straightforward way to assess model accuracy, they have limitations. The **.predict()** method does not provide insights into the confidence or probability associated with each prediction. Similarly, the **accuracy_score()** is highly sensitive to imbalanced datasets, which can lead to misleadingly high or low accuracy readings depending on the distribution of the classes in the dataset.

3. **ROC AUC Score Assessment**: The ROC AUC score is an effective tool for evaluating the binary classification capabilities of a model. However, it can only be used on binary classification tasks and does not extend to multi-class classification scenarios. Additionally, while it provides a good overview of model performance, it does not detail specific areas of misclassification.

4. **Classification Report Analysis**: The **classification_report()** provides a detailed breakdown of the model's performance in terms of precision, recall, and F1-score. However, it is limited to binary classification and might not fully capture the nuances in datasets with a more complex or imbalanced class structure.

5. **Mean Square Error (MSE) and Root Mean Squared Error (RMSE)**: MSE and RMSE are useful for understanding the average error in predictions, but they can be sensitive to outliers. Also, RMSE's interpretation can be easier for non-technical audiences, but it may not always provide a clear indication of the model's performance in classification tasks.

6. **Confusion Matrix**: A confusion matrix offers a visual and quantitative way to understand the performance of a classification model. However, like other techniques listed, it is limited to binary classification and does not convey the nuances of model performance in multi-class scenarios.

7. **Probability Estimates with .predict_proba()**: This function provides a probability estimate for each class in binary classification, offering insight into the model's confidence levels. However, it can be computationally intensive, especially with large datasets, and may not always offer clear actionable insights, particularly in cases of borderline predictions.

8. **Model Interpretation with Coefficients**: Attempting to interpret the model with **trained_model.coef_** coefficients can be challenging due to hardware limitations. This method, while insightful, may not be feasible for large-scale models or datasets, as it requires significant computational resources.

In conclusion, while these evaluation techniques are crucial for assessing the Spam Detection Model, each has specific limitations that must be considered. Understanding these constraints is vital for correctly interpreting the model's performance and for making informed decisions on its application and potential improvements.

Dataset Limitation: The model's training is based on data from 2007, which may not fully represent the evolving nature of SPAM. SPAM tactics and content have likely evolved since then, which could affect the model's effectiveness against current SPAM.

Proposed Actionable Items

1. Deployment: Given the high accuracy and effectiveness of this Logistical Regression Model trained on labeled SPAM(1)/HAM(0) emails, it is recommended to deploy this model in a controlled environment. This will allow for further evaluations of its performance with current email data. This would involve monitoring its classification accuracy in real-time and adjusting thresholds or indicate if we need to gather more data to improve the model.

2. Model Updating and Fine-Tuning: To address the dataset limitation of being an older from 2007 dataset, future studies could focus on continuously updating the model with newer SPAM(1) and HAM(0) datasets. This would help the model adapt to the evolving nature of SPAM and maintain its high accuracy over time.
3. Algorithm Comparison: Exploring and comparing other machine learning algorithms or deep learning models such as Neural Networks for example, might yield even better results or efficiency. Studies could focus on comparing different models to understand which ones are most effective for SPAM(1) detection in various contexts, including different languages, email formats, and evolving SPAM(1) tactics.

Benefits of Study

The study's benefits are highlighted by its quantitative outcomes, offering both immediate usability and a strategic roadmap for future improvements. The logistic regression model demonstrates an exceptional accuracy rate of approximately 98.96%, making it highly reliable for immediate deployment in spam detection. This is crucial in mitigating security risks associated with spam emails.

Furthermore, the balanced dataset, consisting of 47.38% HAM and 52.62% SPAM, underscores the model's robustness, ensuring minimal bias in spam identification. The high AUC score of 98.2% further validates the model’s effectiveness in distinguishing between SPAM and HAM with a low misclassification rate.

In addition to these immediate benefits, the study lays out a two-part future strategy. Firstly, to address the evolving nature of spam, it is recommended to continuously update the model with modern spam datasets, enhancing its relevance and accuracy over time. Secondly, exploring advanced machine learning models, such as neural networks, could provide even more sophisticated spam detection capabilities.

This dual approach not only capitalizes on the current model's strengths but also charts a path for ongoing improvement, adapting to the ever-changing landscape of email threats.