Analysis of N-Grams Model for Smishing Detection

**Introduction**

Smishing, a type of phishing that occurs through SMS messages, poses a significant cyber security threat, making an effective detection system is essential for reducing its impact. Various methods in the field of artificial intelligence can be used to combat smishing attacks, and N-grams are particularly useful for grouping words together to help detect these threats. This report presents an analysis of different N-gram models and their effectiveness in accurately identifying smishing messages.

**Literature Review**

Previous research highlights the use of natural language processing techniques and machine learning algorithms to identify spam messages. N-grams, a method of tokenising text into sequences of words or characters, has proven effective in many text classification tasks (Van Otten 2023). N-grams involve breaking down text into smaller units, this helps understanding word usage and meaning. For example, considering the sentence “The quick brown fox jumps over the lazy dog,” a trigram (N=3) analysis can generate sequences like “The quick brown,” “quick brown fox,” and “brown fox jumps”. These sequences can then be used in various natural language processing tasks such as text classification and language modelling by counting the frequency of each N-gram across a dataset.

Much of the literature suggests that N-grams should be implemented in Python using the Natural Language Toolkit (NLTK). NLTK provides tools for creating and manipulating N-grams efficiently, making it a popular choice for text analysis tasks. For instance, using the ‘ngrams()’ function in NLTK, one can generate bigrams or trigrams from a list of words, which can then be utilised in machine learning models to classify text or perform other natural language processing tasks (NLTK Project 2024).

**Advantages and Disadvantages of N-grams**

N-grams offer several advantages in natural language processing. They efficiently capture local context, which aids in understanding how words and phrases are used. Their flexibility allows them to be applied to a wide range of data sets, from single characters to entire phrases. Additionally, N-grams can be easily computed and stored, making them suitable for large-scale natural language processing tasks. They are also effective as features in machine learning models, enhancing performance in tasks like text classification and information retrieval.

However, N-grams have limitations. While they capture local context well, they may fail to capture broader semantic meanings or discourse-level context. As N increases, the data becomes sparser, leading to challenges in accurately representing all possible N-gram combinations. This data sparsity can result in overfitting, where the model performs well on training data but poorly on unseen data. Furthermore, N-grams lack semantic understanding, meaning they may misinterpret the intent behind words or phrases, leading to errors in some tasks.

**Sample Dataset Description**

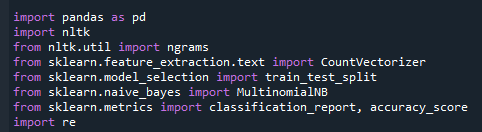
The dataset used consists of SMS messages labeled as either 'ham' (non-spam) or 'spam'. Here are a few examples from the dataset contained in a CSV file:

|  |  |
| --- | --- |
| **Label** | **Message** |
| Ham | Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat... |
| Ham | Ok lar... Joking wif u oni... |
| Spam | Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's |
| Ham | Nah I don't think he goes to usf, he lives around here though |
| Spam | FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, å£1.50 to rcv |

The dataset comprises a mix of casual conversational messages and promotional or scam messages often indicative of smishing.

**How the Program Works**

Importing the Required Libraries



pandas: Used for data manipulation and handling. It allows easy loading, processing, and analysis of structured data like the sample CSV file used in this program.

nltk (Natural Language Toolkit): A library for text processing in Python. It provides tools for tokenisation, stemming, and other Natural Language Processing tasks. Specifically, ngrams from nltk.util is used to generate n-grams (sequence of n words).

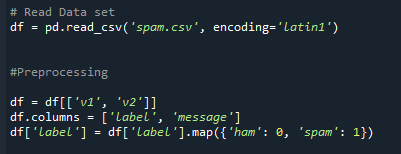
sklearn.feature\_extraction.text.CountVectorizer: Converts a collection of text documents to a matrix of token counts, transforming the text data into a numerical format suitable for machine learning models.

sklearn.model\_selection.train\_test\_split: Splits the dataset into training and testing subsets. It is crucial for evaluating the model's performance on unseen data.

sklearn.naive\_bayes.MultinomialNB: Implements the Multinomial Naive Bayes classifier, which is effective for discrete features such as word counts.

sklearn.metrics: Provides functions to measure the accuracy of models and generate a classification report.

Data Loading and Preprocessing



Data Loading: The dataset is loaded using pandas.read\_csv() with the appropriate encoding. This dataset contains text messages labelled as either 'ham' or 'spam'.

Data Selection and Renaming: Only relevant columns ('v1' for label and 'v2' for message) are selected and renamed for clarity.

Label Encoding: The labels are mapped to numerical values (0 for 'ham' and 1 for 'spam') to facilitate the model training process.

Text Preprocessing and N-gram Extraction

A computer screen shot of text

Description automatically generated

Tokenisation and Cleaning: The tokenize\_text function cleans the text by removing punctuation and tokenises it into individual words. Tokenisation is essential for breaking down the text into manageable units.

N-gram Generation: The extract\_ngrams function generates n-grams from the tokenised text. In the case, with n=1, the program extracts unigrams.

Message to N-grams Conversion: message\_to\_ngrams converts messages into a format suitable for n-gram analysis by creating space-separated n-grams.

Vectorisation and Model Training

A computer screen shot of a program

Description automatically generated

Vectorisation: The CountVectorizer converts the n-gram messages into numerical features (word frequency vectors) that the classifier can process.

Data Splitting: The dataset is split into training (80%) and testing (20%) sets. This separation is vital for evaluating the model's generalisation capability.

Model Training: The MultinomialNB classifier is trained using the training data. It learns the relationship between word occurrences and message labels (ham or spam).

Model Evaluation

A computer code with white text

Description automatically generated

Prediction and Evaluation: The trained model predicts the labels for the test set. The accuracy and classification report provide insights into the model's performance, including precision, recall, and F1-score for both ham and spam classes.

**Methodology**

A dataset of SMS messages labelled as 'ham' (legitimate) or 'spam' (smishing) was used to train and test the models. The text was pre-processed using standard Natural Language Processing techniques, including tokenisation, lowercasing, and removal of stop words. Three N-gram models (N=1, N=2, N=3) were implemented using the Natural Language Toolkit (NLTK). The models were evaluated based on training accuracy, test accuracy, and the detailed classification metrics of precision, recall, F1-score, and support.

**Aim of the Study and Experimental Results**

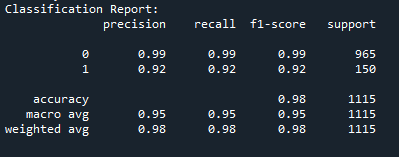
The aim of this study was to test the effectiveness of different types of N-grams in detecting smishing messages. Messages were graded as 0 for 'ham' (legitimate) and 1 for 'spam' (smishing). The performance of unigrams (N=1), bigrams (N=2), and trigrams (N=3) were evaluated using a classification report that includes precision, recall, F1-score, and support metrics.

Unigrams (N=1) showed the highest accuracy and were the most effective in detecting smishing messages, with a balance of high precision and recall. As the size of N-grams increased to bigrams (N=2) and trigrams (N=3), the accuracy decreased, and the classification became less effective. Due to this declining accuracy trend, it was determined that testing higher N-grams (e.g., N=4, N=5, etc.) would not be beneficial, as the performance continued to degrade with increasing N-gram size.

**Unigrams (N=1) Results:**

Training Accuracy: 99.15%

Test Accuracy: 97.85%



Precision: Precision for ham (0.99) indicates that almost all messages classified as ham are non-spam. The precision for spam (0.92) shows that 92% of the messages classified as spam are actually smishing messages. High precision means fewer false positives, which is crucial to avoid marking legitimate messages as spam.

Recall: Recall for ham (0.99) suggests that nearly all non-spam messages were correctly identified. The recall for spam (0.92) means that 92% of the actual spam messages were correctly detected. High recall indicates the model's effectiveness in capturing smishing messages, minimising false negatives.

F1-score: This metric balances precision and recall. With both spam and ham having an F1-score close to 0.99 and 0.92, respectively, the unigram model shows a strong balance, making it highly effective.

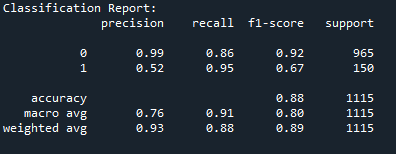
Support: The support indicates the number of actual occurrences of each class in the dataset. The ham class has a high support (965), which can influence the overall accuracy due to class imbalance. However, the spam class with 150 occurrences still shows good performance.

The unigram model's high precision and recall for both classes indicate it effectively detects smishing messages while minimising misclassifications. This balance makes unigrams the most effective type of N-gram for detecting smishing.

**Bigrams (N=2) Results:**

Training Accuracy: 99.98%

Test Accuracy: 87.62%



Precision: While ham maintains high precision (0.99), spam precision drops to 0.52, meaning nearly half of the messages flagged as spam were actually non-spam. This high rate of false positives suggests overfitting, where the model learns patterns that do not generalise well to new data.

Recall: The recall for ham decreases (0.86), showing that some non-spam messages are misclassified as spam. However, recall for spam is very high (0.95), indicating that most spam messages are detected.

F1-score: The F1-score for ham (0.92) remains high due to good precision and recall, but the spam F1-score drops to 0.67, reflecting the lower precision. This disparity indicates that the bigram model struggles with accurately identifying spam messages.

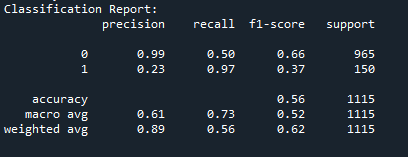
Support: Like unigrams, the support values (965 for ham and 150 for spam) show that while ham still dominates, the relatively low performance on spam indicates issues with generalisability for this class.

The bigram model's high recall but low precision for spam messages indicates it catches most spam but at the cost of many false alarms. This trade-off is less desirable in a smishing detection context, where precision is crucial to prevent legitimate messages from being blocked.

**Trigrams (N=3) Results:**

Training Accuracy: 99.98%

Test Accuracy: 56.05%



Precision: The precision for ham remains high (0.99), but spam precision is extremely low (0.23), indicating that the majority of messages classified as spam are actually not. This very high rate of false positives suggests severe overfitting and poor generalisation to the test data.

Recall: The recall for ham drops to 0.50, meaning half of the non-spam messages are being misclassified as spam. For spam, the recall is high (0.97), indicating almost all actual spam messages are detected. This high recall suggests the trigram model is over-sensitive to spam features.

F1-score: The ham F1-score drops to 0.66, and for spam, it is 0.37, indicating poor performance. The low F1-score for spam shows that while most spam is caught, the model's precision is so low that many legitimate messages are falsely flagged.

Support: As with the other models, ham messages are more prevalent. The low performance for ham despite its high support indicates that trigrams introduce too much complexity, leading to overfitting.

The trigram model's low precision and F1-score, coupled with its high recall, suggest it is overly complex and sensitive, making it inefficient for real-world applications where minimising false positives is crucial.

**Discussion**

The findings indicate that unigrams (N=1) provide the best balance of precision and recall, making them the most effective for smishing detection. Bigrams and trigrams, while capturing more context through the use of larger N-grams, tend to overfit the data, resulting in a higher rate of false positives. These results suggest that simplicity (unigrams) is preferable for text classification tasks, particularly in areas like smishing detection where false positives can significantly impact user experience.

**Conclusion**

This study demonstrates that unigrams (N=1) are the most effective N-gram model for detecting smishing messages, achieving high precision and recall and minimising false positives. While bigrams and trigrams offer more complex pattern recognition, their potential to overfit and produce false positives makes them less suitable for smishing detection. Future work could explore combining N-gram models with other text features or using advanced deep learning techniques to further improve detection accuracy.

**Recommendations**

Implement unigrams as the primary N-gram model for smishing detection systems due to their balanced performance.

Further research could focus on enhancing model performance by integrating other features like message metadata, sender information, or using ensemble methods to combine the strengths of different models.

**Reference List**

NTLK Project (19 August 2024), [‘Natural Language Toolkit Documentation’](https://www.nltk.org/), NTLK Project, accessed 24 August 2024.

Van Otten N (5 April 2023) [‘N-grams Made Simple & How To Implement In Python (NLTK)’](https://spotintelligence.com/2023/04/05/n-grams/#How_to_implement_n-grams_in_Python_with_NLTK), Spot Intelligence, accessed 24 August 2024.