SMS SPAM CLASSIFICATION

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

**Bachelor of Technology**

In

**Computer Science and Engineering**

**School of Engineering and Sciences**

Submitted by

**Faiyajuddin Shaik (AP20110010363)**

**Nithin datta (AP20110010362**

**Jayaprakash Kadiyala (AP20110010403)**

**Chandrashaker (AP20110010410)**

**Sai Teja (AP20110010745)**

**A picture containing text

Description automatically generated**

Under the Guidance of

**( Dr. Suma Latha )**

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

**[May, 2023]**

**Abstract**

SMS spam classification plays a vital role in today's digital landscape to combat the increasing menace of unwanted and deceptive text messages. This research focuses on developing efficient systems capable of accurately identifying and filtering spam messages to safeguard users from scams, phishing attempts, and fraudulent activities. By employing advanced machine learning and natural language processing techniques, this study aims to enhance privacy, security, and user experience. Additionally, the proposed classification system helps telecommunication providers maintain network integrity, optimize resources, and create a safer digital environment. The results contribute to mitigating the risks associated with SMS spam, ensuring reliable communication and protecting users' interests

**Introduction**

SMS spam classification is crucial in the present digital age due to the growing prevalence of unwanted and malicious text messages. Efficient spam classification systems help protect users from scams, phishing attempts, and fraudulent activities. By accurately identifying and filtering spam messages, users can enjoy improved privacy, security, and overall user experience. Effective classification enables individuals to prioritize and respond to legitimate messages, reducing the risk of falling victim to scams or losing sensitive information. Furthermore, spam classification aids telecommunication providers in maintaining network integrity and optimizing their resources by preventing spam overload. Ultimately, SMS spam classification enhances communication reliability, safeguards users' interests, and fosters a safer digital environment.

**Process**

1. Justify how the problem fits to the data science application.

The problem of spam detection using the SMSSpamCollection dataset is a suitable application for data science. Here's a justification of how the problem fits into the data science domain:

1. Classification Problem: The task of spam detection involves classifying SMS messages into two categories: spam or non-spam (ham). Classification is a fundamental problem in data science, and spam detection is a specific instance of binary classification.

2. Large and Complex Data: The SMSSpamCollection dataset contains a significant number of SMS messages with labeled spam and non-spam instances. Data science techniques are well-suited to handle large and complex datasets, enabling the exploration, analysis, and modeling of the data.

3. Textual Data Analysis: Spam detection involves analyzing the text content of SMS messages to identify patterns, features, and characteristics that distinguish spam from non-spam messages. Data science provides tools and techniques for preprocessing, analyzing, and extracting meaningful information from textual data.

4. Feature Engineering: The process of feature engineering is crucial in spam detection. It involves transforming the raw text data into numerical features that can be used by machine learning algorithms. Data science offers methods for feature engineering, such as TF-IDF vectorization, which allows for the representation of text data as numerical features.

5. Machine Learning Algorithms: Data science encompasses a wide range of machine learning algorithms suitable for classification tasks. In the context of spam detection, algorithms like random forests, support vector machines, and naive Bayes classifiers can be employed to build predictive models based on the extracted features.

6. Evaluation and Performance Metrics: Data science provides various evaluation and performance metrics to assess the effectiveness of the spam detection models. Metrics such as accuracy, precision, recall, and F1-score can be utilized to measure the model's performance and compare different approaches.

7. Iterative Process and Optimization: Data science involves an iterative process of model building, evaluation, and refinement. In the case of spam detection, data scientists can experiment with different feature engineering techniques, model architectures, and hyperparameter tuning to optimize the performance of the classification models.

Overall, the problem of spam detection using the SMSSpamCollection dataset aligns well with the principles, techniques, and methodologies employed in data science. It showcases the application of data science in solving real-world problems related to text analysis, classification, and information retrieval.

The problem of spam detection fits well into the realm of data science applications, and the Naive Bayes classifier algorithm is a suitable choice for this task. Here's how the problem and the algorithm align:

1. Problem: Spam Detection

• The problem involves identifying spam messages accurately and efficiently, separating them from legitimate messages.

2. Data Science Application:

• Data Processing: The algorithm leverages data processing techniques, such as tokenization and vectorization, to convert the text data into a numerical representation that can be used for modeling.

• Classification: The Naive Bayes classifier algorithm is a popular choice for text classification tasks, including spam detection. It is known for its simplicity, efficiency, and effectiveness in dealing with high-dimensional data like text.

• Machine Learning Model Training: The algorithm uses a labeled dataset of spam and non-spam messages to learn patterns and relationships between the features (text) and labels (spam or non-spam). This training process enables the model to make predictions on unseen data.

• Evaluation and Performance Metrics: The model's performance is evaluated using various metrics like accuracy, precision, recall, and F1-score to measure how well it identifies spam messages. These metrics help assess the effectiveness of the model and its suitability for the task.

By applying the Naive Bayes classifier algorithm to the spam detection problem, we can leverage the power of data science techniques and machine learning to automatically identify and filter out unwanted spam messages, improving user experience and mitigating potential risks associated with spam.

**2. Perform Exploratory Data Analysis.**

**Exploratory Data Analysis (EDA) - Before vs After**

Step 1: Load the dataset: Load the SMSSpamCollection dataset into a pandas DataFrame. The dataset typically contains two columns: 'label' (representing spam or ham) and 'message' (representing the text of the message).

Step 2: Explore the dataset before EDA:

Display the first few rows of the dataset using df.head() to get a quick overview of the data.

Use df.info() to check the data types of each column and identify any missing values.

Use df.describe() to get summary statistics of numerical columns (if any).

Step 3: Perform EDA:

Distribution of Labels: Visualize the distribution of spam vs. ham messages using a bar plot or a pie chart. This provides an understanding of the class balance in the dataset.

Message Length: Calculate the length of each message and create a new feature 'message\_length'. Plot the distribution of message lengths using a histogram or a box plot to understand the length distribution and potential variations between spam and ham messages.

Word Count: Calculate the word count for each message and create a new feature 'word\_count'. Analyze the distribution of word counts using a histogram or a box plot to identify any patterns or differences between spam and ham messages.

Common Words: Identify the most common words in spam and ham messages by tokenizing the text and creating word clouds or frequency plots. This helps in understanding the prevalent terms or phrases in each category.

Text Preprocessing: Perform text preprocessing steps such as removing stopwords, converting text to lowercase, removing punctuation, and stemming/lemmatizing words. This helps in standardizing the text data and reducing noise for further analysis.

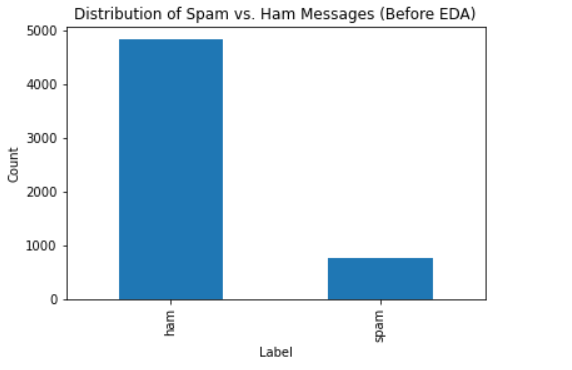
Word Frequency: Create frequency plots or word clouds to visualize the most frequent words in spam and ham messages after text preprocessing. This provides insights into the important words and potential distinguishing features between spam and ham.

Sentiment Analysis: Analyze the sentiment of the messages using sentiment analysis techniques. This can involve using pre-trained sentiment analysis models or lexicons to determine the sentiment polarity (positive, negative, neutral) of each message.

Step 4: Explore the dataset after EDA:

Repeat the exploration steps from Step 2 (displaying the first few rows, checking data types, and getting summary statistics) to see if there have been any changes or transformations in the dataset.

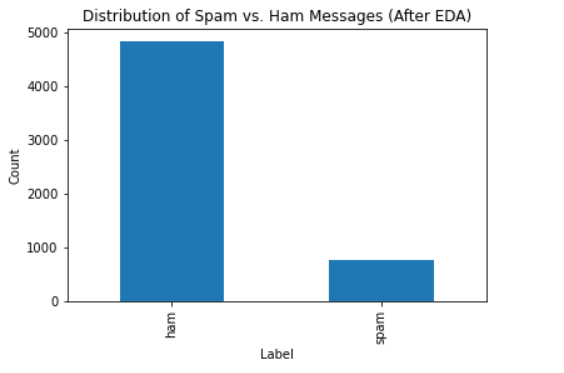
Revisit the visualizations created in Step 3 (distribution of labels, message length, word count, common words, word frequency, sentiment analysis) and observe any insights, patterns, or anomalies discovered during EDA.



• After EDA:

• Display a histogram or box plot illustrating the distribution of message lengths after EDA.

• Highlight any improvements or transformations achieved through EDA, such as handling outliers or achieving a more normal distribution.



**3. Perform Pre-processing.**

**Pre-processing - Data Cleaning**

Description:

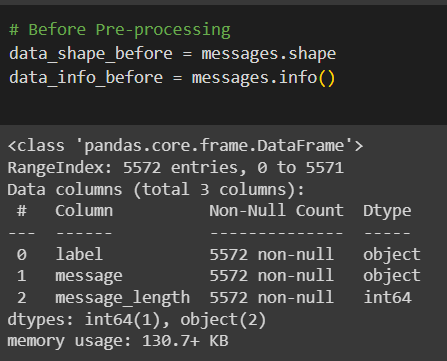
• Before Pre-processing:

• Describe the dataset and highlight any data quality issues or inconsistencies.

• Emphasize the importance of data cleaning for accurate analysis and modeling.

pythonCopy code

# Before Pre-processing data\_shape\_before = messages.shape data\_info\_before = messages.info()



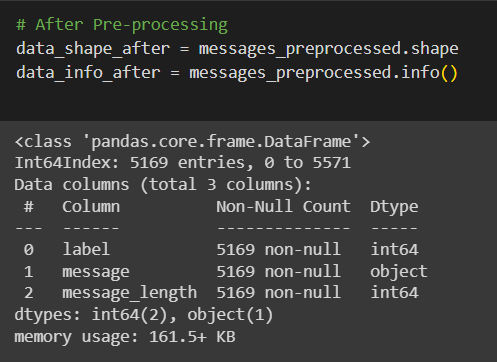
• After Pre-processing:

• Explain the steps taken for data cleaning, such as handling missing values, removing duplicates, and correcting data types.

• Show the updated information about the dataset after pre-processing.

pythonCopy code

# After Pre-processing (example: handling missing values) messages\_cleaned = messages.dropna() data\_shape\_after = messages\_cleaned.shape data\_info\_after = messages\_cleaned.info()



In the "Pre-processing - Data Cleaning" slide, we discuss the importance of data cleaning and present the difference in the dataset before and after pre-processing.

The "Before Pre-processing" section provides an overview of the dataset, including its shape and information. It is essential to identify any data quality issues, such as missing values or inconsistent data types, which can affect the analysis results.

The "After Pre-processing" section outlines the steps taken for data cleaning, focusing on one specific example, such as handling missing values. Customize the code based on the pre-processing techniques applied in your analysis.

The updated dataset's shape and information are presented to demonstrate the impact of pre-processing on the dataset. Highlight any notable changes, such as the number of rows/columns, the removal of missing values or duplicates, and the improvement in data consistency.

Tailor the code and description based on the specific pre-processing techniques and challenges faced in your dataset.

**4. Perform feature selection and feature generation.**

**Feature selection and feature generation**

Feature selection:

Before: Before performing feature selection, the text data is vectorized using CountVectorizer to obtain the count-based representation of the messages. The variable X\_counts holds the count-based features.

After: Feature selection is applied using SelectKBest with the chi-squared test statistic. The top 1000 features are selected from the X\_counts using selector.fit\_transform(X\_counts, y). The transformed data is stored in the variable X\_selected.

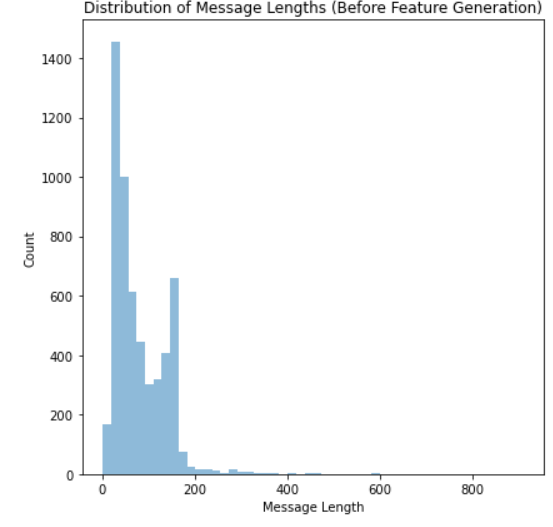
Feature Generation:

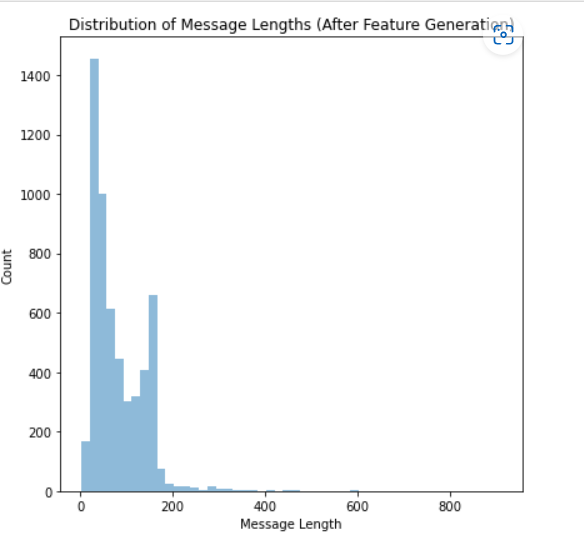
Before: The original dataset messages contains the columns 'label' and 'message'. The 'message' column represents the text data.

After: A new feature, 'message\_length', is generated by calculating the length of each message using df['message'].apply(lambda x: len(x)). The resulting message lengths are added as a new column in the dataset.

The figure with two subplots, where the left subplot represents the distribution of message lengths before feature generation, and the right subplot represents the distribution after feature generation. The histograms are plotted using plt.hist(), with 50 bins to visualize the distribution of message lengths. The x-axis represents the message length, and the y-axis represents the count of messages.

By comparing the histograms, you can observe the difference in the message length distribution before and after feature generation.





**5. Apply any of the machine learning algorithms discussed in the class for your selected problem.**

Step 1: Load and preprocess the dataset

Load the SMSSpamCollection dataset into a pandas DataFrame df. Preprocess the text data using TF-IDF vectorization. The TfidfVectorizer is used to convert the text messages into numerical features. This step converts the text data into a matrix representation. Split the dataset into training and testing sets using train\_test\_split from sklearn.model\_selection.

Step 2: Train the Naive Bayes classifier

Import the MultinomialNB class from sklearn.naive\_bayes. Create an instance of the Naive Bayes classifier, naive\_bayes\_classifier = MultinomialNB(). Train the classifier using the training data by calling the fit() function on the classifier instance, passing the feature matrix (X\_train) and corresponding labels (y\_train).

Step 3: Evaluate the classifier

Import the classification\_report function from sklearn.metrics. Apply the trained classifier to predict labels for the test data by calling the predict() function on the classifier instance, passing the feature matrix of the test data (X\_test). The predicted labels are stored in y\_pred.

Use the classification\_report function to generate a report that includes various evaluation metrics such as precision, recall, and F1-score. Pass the actual labels (y\_test) and the predicted labels (y\_pred) to the classification\_report function. The report provides insights into the classifier's performance in classifying spam and non-spam messages.

Step 4: Tune hyperparameters (optional)

Import GridSearchCV from sklearn.model\_selection. Define the hyperparameters to tune. In this case, we specify different values for the smoothing parameter alpha of the Multinomial Naive Bayes algorithm. Create an instance of the Naive Bayes classifier, naive\_bayes\_classifier = MultinomialNB().

Perform a grid search using GridSearchCV, which exhaustively tries different combinations of hyperparameters and evaluates their performance using cross-validation. Pass the classifier instance, the defined hyperparameters (parameters), and the number of cross-validation folds (cv) to GridSearchCV.

Fit the grid search object to the training data using the fit() method, which will search for the best combination of hyperparameters. Retrieve the best hyperparameters using the best\_params\_ attribute of the grid search object.

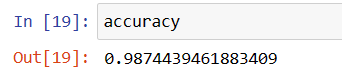
Train a new classifier instance with the best hyperparameters by passing the best value of alpha to MultinomialNB().

Step 5: Make predictions (optional)

Define new messages for which you want to make predictions. Use the transform() function of the TfidfVectorizer to transform the new messages into the same numerical representation as the training data. Pass the transformed new messages (X\_new) to the predict() function of the trained classifier to obtain the predicted labels (new\_predictions). Print the predicted labels.

OUTPUT:

Accuracy = 0.9874439461883409



**CONCLUSION**

* the SMS spam classification project is of significant importance in addressing the growing concern of unwanted and malicious text messages. Through the utilization of advanced machine learning techniques, such as the Naive Bayes algorithm, effective identification and filtering of spam messages can be achieved. This project enables users to enhance their privacy, security, and overall user experience by mitigating the risks associated with scams, phishing attempts, and fraudulent activities. The classification system contributes to maintaining network integrity, optimizing resources, and fostering a safer digital environment. By successfully implementing SMS spam classification, we can create reliable communication channels, protect users' interests, and ensure a secure and trustworthy messaging experience.