

De La Salle University – Dasmariñas
College of Information and Computer Studies
Computer Science Department

**SiFri-Mail: A Pertinent Communication and Categorization
Tool through the E-mail Protocol using Natural Language
Processing and Support Vector Mechanism**

Submitted in Partial Fulfillment of the Requirement for the Degree of
Bachelor of Science in Computer Science

FRIOLO, CHRISTIAN C.

IMPERIAL, LUIS ANTON P.

SICAD, TIMOTHY ALLEN N.

December 03, 2024

Abstract

Natural Language Processing (NLP) has emerged as a powerful tool for analyzing and categorizing text-based data, such as emails. The "Sifri-Mail" project is a proof-of-concept for a cross-platform, Artificial Intelligence (AI)-powered electronic mail (e-mail) client application program that will simplify the daily workflow of users by linking all their email accounts into one place before categorizing them by topic and priority through Natural Language Processing (NLP) and Support Vector Machines (SVM). It seeks to leverage these technologies to develop a platform- and provider-agnostic email categorization tool. By using NLP and SVM, Sifri-Mail aims to automatically categorize emails, enhancing efficiency and ensuring that important messages are prioritized.

Keywords

categorization, email, email client, natural language processing, social networks, support vector mechanism, language model, artificial intelligence.

Chapter 1: Introduction

1.1 Project Context

In today's digital landscape, email has become an essential communication tool, with billions of messages exchanged daily. However, the increasing volume of emails has led to challenges in effectively managing and categorizing incoming messages. This issue is particularly problematic in professional settings, where efficient communication is crucial for maintaining productivity and ensuring that important messages are not overlooked. Studies indicate that the number of emails sent and received globally is expected to exceed 347 billion per day by 2023, underscoring the need for more advanced solutions to manage this communication flow.

To address these challenges, Natural Language Processing (NLP) has emerged as a powerful tool for analyzing and categorizing text-based data, such as emails. NLP techniques enable systems to comprehend and process human language with a high degree of accuracy, making it possible to categorize emails based on their content rather than just keywords. When combined with Support Vector Machines (SVM), which are effective for text classification, these technologies can significantly improve the accuracy and efficiency of email categorization [3].

The "Sifri-Mail" project seeks to leverage these technologies to develop a platform- and provider-agnostic email categorization tool. By using NLP and SVM, Sifri-Mail aims to automatically categorize emails, enhancing efficiency and ensuring that important messages are

prioritized. This approach is particularly relevant in today's fast-paced digital environment, where efficient communication is key to organizational success. [1]

1.2 Purpose and Description

The purpose of the SiFri-Mail project is to create an intelligent tool that streamlines email management by automatically categorizing emails based on their content. By employing advanced technologies like Natural Language Processing (NLP) and Support Vector Machines (SVM), the tool is capable of understanding and analyzing the text of incoming emails with a high degree of accuracy. This ensures that important messages are promptly identified and prioritized, while less relevant ones are effectively organized. The tool aims to improve the overall efficiency of managing large volumes of emails, reducing the time and effort required for manual sorting, and helping users maintain a more organized and functional inbox. This enhanced email management will contribute to better communication and productivity, especially in environments where the timely processing of information is critical.

1.3 Statement of the Problem

The main problem is the inefficiency and inaccuracy of current email management systems in handling the growing volume and complexity of email communications. Traditional methods, such as manual sorting and keyword-based filtering, often fail to effectively manage the diverse and voluminous nature of modern email content. This inadequacy leads to important emails being overlooked, miscategorized, or lost in the clutter, which can result in missed opportunities, decreased productivity, and communication breakdowns.

- How might the system handle ambiguous or contextually complex emails that could lead to incorrect categorization?
- What challenges could arise in ensuring that the categorization system adapts to evolving email content and user needs over time?
- How can the system balance accuracy and speed in processing large volumes of emails without compromising on either?

1.4 Research Objective

The research objectives for SiFri-Mail aim to address the growing complexities of email management in both personal and professional environments. With the daily volume of emails reaching unprecedented levels, traditional methods of email organization, such as manual sorting and basic keyword filtering, no longer suffice. This project seeks to enhance email management by leveraging advanced technologies, specifically Natural Language Processing (NLP) and Support Vector Mechanism (SVM), to create an intelligent, automated system capable of classifying and prioritizing emails based on their content. By doing so, SiFri-Mail strives to improve productivity, reduce the time spent on email management, and ensure that critical communications are not lost in the clutter of inboxes.

The objectives outlined below encompass both the general goal of developing a robust email categorization tool and the specific technical and functional milestones that will guide the development and evaluation of the system.

1.4.1 General Objectives

The general objective of the SiFri-Mail project is to develop a pertinent communication and categorization tool through the email protocol using natural language processing and support vector mechanism.

1.4.2 Specific Objectives

1. To design and implement a machine learning-based email categorization system using NLP and SVM that automatically sorts emails according to their content and relevance.
2. To evaluate the accuracy of the email categorization system by comparing its performance against traditional keyword-based filtering methods in real-world scenarios.
3. To ensure the scalability of the system in handling large volumes of emails without compromising processing speed or accuracy.
4. To adapt the system for evolving email content and user needs through continuous learning and updates to the classification model, ensuring long-term relevance and accuracy.
5. To conduct user testing and feedback collection to refine the tool's user interface and functionality, ensuring it meets the practical needs of its users across different platforms.
6. To minimize the manual effort involved in email management by providing users with customizable categorization options, enabling personalized organization and prioritization of emails.

1.5 Significance of the Study

Due to the lack of an intuitive, affordable, AI-powered communication tool on the market, this study aims to simplify daily organizational workflows by providing our target customer base with an easy-to-use email app that incorporates natural language processing technology to minimize the busywork of categorization and sorting.

1.6 Scope and Delimitation

The scope of this research covers the analysis of email categorization using Natural Language Processing (NLP) techniques with a specific focus on sentiment analysis. The primary objective is to assess the effectiveness of sentiment-based email classification to improve user prioritization and management of emails. This involves categorizing emails by urgency, relevance, and context using a support vector mechanism to train the sentiment analysis model.

Data collection for this research includes testing by 20 to 50 respondents, who will use the application and provide feedback. The feedback will be gathered through structured online surveys hosted on Google Forms, focusing on user experience, accuracy of email categorization, and overall satisfaction with the tool. Questions will cover the clarity of categorization, perceived accuracy, ease of use, and areas for improvement.

Delimitation of this study excludes the technical specifications of the application, such as programming languages and platforms used. The research will not explore technical development or backend processing. Additionally, email content data will be anonymized to preserve privacy, and the study will avoid any long-term monitoring of user email habits post-research. Questions unrelated to sentiment accuracy, such as in-depth technical feedback on interface design, will not be included in the survey.

Chapter 2: Discussion of Algorithms

2.1 Explanation

2.1.1 Merge Sort

Takes an input of an array of objects (the email data) received from the backend email API and manipulates the order of the elements in such a way wherein the array is ordered based on one of its specified attributes.

This is done by the process below:

- The array of objects is divided in half recursively until it cannot be divided any further.
- A specified attribute (date, subject, etc.) of the objects present in each subarray are individually evaluated then sorted.
- The now sorted subarrays are merged back together in sorted order. This process repeats until all subarrays are merged back into the complete, sorted list.

Pseudocode

```
FUNCTION merge(array, left, mid, right)
    n1 = mid - left + 1
    n2 = right - mid

    l_arr = [0] * n1
    r_arr = [0] * n2

    For i in range(n1)
        l_arr[i] = array[left + i]
    For j in range(n2)
        r_arr[j] = array[mid + 1 + j]

    i, j, k = 0, 0, left

    While i < n1 and j < n2
        If l_arr[i] <= r_arr[j] Then
            array[k] = l_arr[i]
            i += 1
        Else
            array[k] = r_arr[j]
            j += 1
        k += 1
```

```

While i < n1
    array[k] = l_arr[i]
    i += 1
    k += 1

While j < n2
    array[k] = r_arr[j]
    j += 1
    k += 1

FUNCTION mergeSort(array, left, right)
    If left < right Then
        mid = (left + right) / 2 Rounded Down

        mergeSort(array, left, mid)
        mergeSort(array, mid + 1, right)
        merge(array, left, mid, right)

```

2.1.2 Filter by Search

Takes an input of an array of objects (the email data) received from the backend email API and evaluates the objects inside it. The algorithm checks to see if certain attributes have partial or full matches to a user specified string, and returns a version of the array with objects that meet that match.

Pseudocode

```
FUNCTION filter(array, query)
    filtered_array = []

    For i in array
        If i matches query Then
            append i to filtered_array

    return filtered_array
```

2.1.3 Email Classifier

Takes a string input (the email contents), passes it to a trained SVC (Support Vector Classifier) model which evaluates the string and assigns it a label, which it returns. An SVC model is a kind of algorithm that finds the optimal hyperplane to separate data points by different classes. It takes the features of the input and predicts which decision boundary (class) it falls under.

Pseudocode

```
data = array of email data
array_labels = []

For i in data
    // pre trained classifier model
    label = svm_classifier.predict(i.body_of_email)

    array_labels.append(label)
```

2.2 Sample Run

Merge Sort

```
// Merge sort algorithm for sorting emails
const mergeSort = useCallback((array, sortBy) => {
  if (array.length <= 1) return array; // Base case for recursion

  const middle = Math.floor(array.length / 2);
  const left = mergeSort(array.slice(0, middle), sortBy); // Sort left half
  const right = mergeSort(array.slice(middle), sortBy); // Sort right half

  return merge(left, right, sortBy); // Merge sorted halves
}, []); // Empty dependency ensures stable reference

// Helper function to merge two sorted arrays
const merge = (left, right, sortBy) => {
  let resultArray = [];
  let leftIndex = 0;
  let rightIndex = 0;

  while (leftIndex < left.length && rightIndex < right.length) {
    if (sortBy === "sender") {
      if (left[leftIndex].sender.localeCompare(right[rightIndex].sender) < 0) {
        resultArray.push(left[leftIndex]); // Push from left array
        leftIndex++;
      } else {
        resultArray.push(right[rightIndex]); // Push from right array
        rightIndex++;
      }
    } else if (sortBy === "subject") {
      if (left[leftIndex].subject.localeCompare(right[rightIndex].subject) < 0) {
        resultArray.push(left[leftIndex]); // Push from left array
        leftIndex++;
      } else {
        resultArray.push(right[rightIndex]); // Push from right array
        rightIndex++;
      }
    } else {
      if (new Date(left[leftIndex].date) > new Date(right[rightIndex].date)) { // Sort by date
        resultArray.push(left[leftIndex]);
        leftIndex++;
      } else {
        resultArray.push(right[rightIndex]);
        rightIndex++;
      }
    }
  }

  return resultArray
    .concat(left.slice(leftIndex)) // Append remaining left elements
    .concat(right.slice(rightIndex)); // Append remaining right elements
};
```

Filter by Search

```
// Filter and sort emails based on search query, sorting criteria, and active category
useEffect(() => {
  const updatedEmails = emails.filter(
    (email) =>
      email.subject.toLowerCase().includes(searchQuery.toLowerCase()) ||
      email.sender.toLowerCase().includes(searchQuery.toLowerCase()) // Match search query
  );

  setFilteredEmails(mergeSort(updatedEmails, sortCriteria)); // Sort filtered emails

  if (activeCategory !== "Inbox") {
    setFilteredEmails((prevFilteredEmails) =>
      prevFilteredEmails.filter((email) => email.classification.includes(activeCategory)) // Filter
    );
  }
}, [searchQuery, sortCriteria, emails, activeCategory, mergeSort]); // Rerun on dependency change
```

Email Classifier

```
class EmailClassifier:
    def __init__(self):
        """Initialize the email classifier by loading the pre-trained model and vectorizer."""
        print("Loading model and vectorizer...")
        try:
            # Paths to the pre-trained model and vectorizer
            model_path = 'C:/Users/CHRISTIAN/OneDrive/Desktop/Website_log/backend/classifier/model/svm
            vectorizer_path = 'C:/Users/CHRISTIAN/OneDrive/Desktop/Website_log/backend/classifier/mode

            # Debugging information to verify the current working directory
            print(f"Current working directory: {os.getcwd()}")
            if not os.path.exists(model_path):
                print(f"Model file not found: {model_path}") # Log if model file is missing
            if not os.path.exists(vectorizer_path):
                print(f"Vectorizer file not found: {vectorizer_path}") # Log if vectorizer file is mi

            # Load the model and vectorizer
            self.model = joblib.load(model_path)
            self.vectorizer = joblib.load(vectorizer_path)
            print("Model and vectorizer loaded successfully.")
        except Exception as e:
            print(f"Error loading model or vectorizer: {e}") # Log any errors encountered during load

    def classify(self, emails):
        """
        Classify a list of emails into predefined categories.

        Args:
            emails (list): List of email bodies as strings.

        Returns:
            list: Predicted labels for each email.
        """
        print("Classifying emails...")
        try:
            # Extract plain text from email bodies
            email_texts = [self.extract_text(email) for email in emails]
            email_tfidf = self.vectorizer.transform(email_texts) # Vectorize the cleaned email text
            predictions = self.model.predict(email_tfidf) # Predict email categories

            # Map numeric predictions to corresponding labels
            labels = ["Important", "Spam", "Drafts", "Inbox"]
            labeled_predictions = [labels[p] for p in predictions]

            print(f"Labeled Predictions: {labeled_predictions}") # Log the predictions
            return labeled_predictions
        except Exception as e:
            print(f"Error during classification: {e}") # Log any errors during classification
            return []
```

Output


≡

Search emails... Sort by Date ▼

Document shared with you: "S-CSPC315 Final Project Friolo Imperial Sicad"
From: drive-shares-dm-noreply@google.com
Classification: Important
Date: December 2, 2024 at 09:10 PM

Timothy Sicad sent a message to the group conversation "S-CSSE321 - PROJECT".
From: messages@facebookmail.com
Classification: Important
Date: December 2, 2024 at 08:42 PM

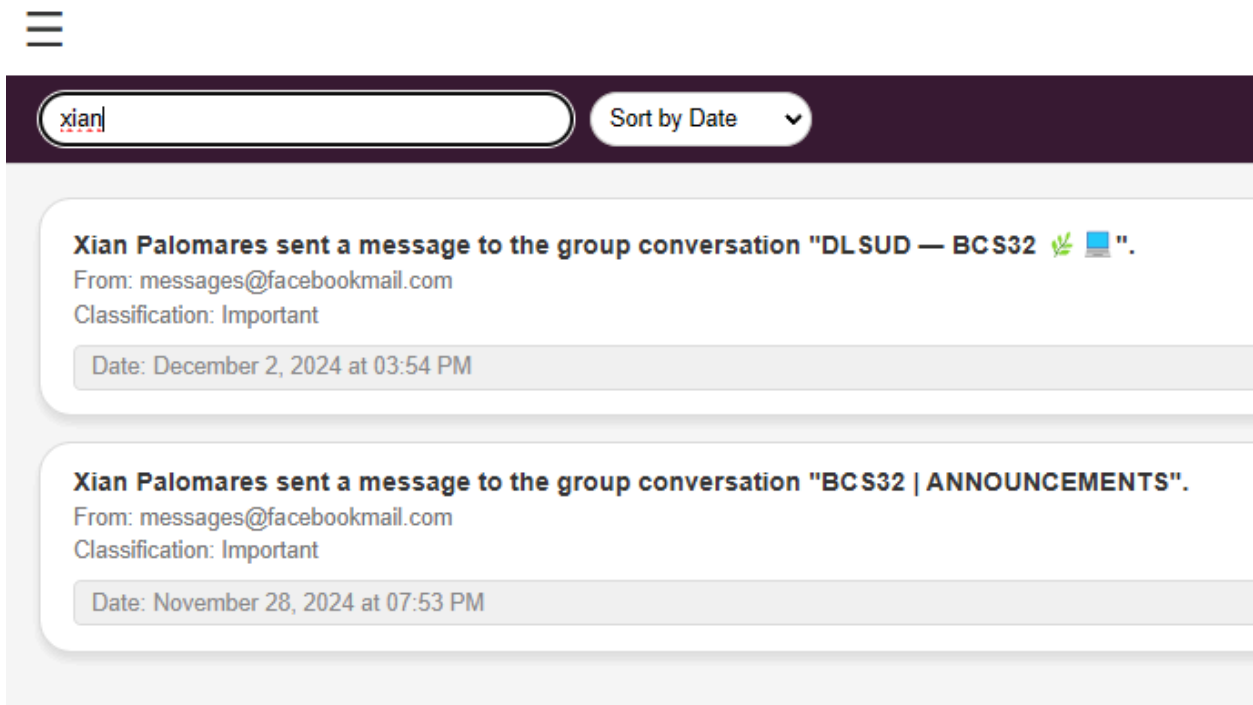
Final Day - Packages Closing Soon
From: sensei@smilenihongo.com
Classification: Important
Date: December 2, 2024 at 08:03 PM

 **Surprise! Save 70% during Avast's Cyber Monday**
From: notification@emails.avast.com
Classification: Important
Date: December 2, 2024 at 07:42 PM

Make Monday count with this US\$99¢ deal
From: email@washingtonpost.com
Classification: Important
Date: December 2, 2024 at 07:04 PM

[Register Now] Incident Response of the Network – Navigating Cyber Crises
From: netacademail@external.cisco.com
Classification: Important

1 2 3 4 5 6 7 8 9 10



2.3 Relevance to Algorithm

Due to the nature of an email client, the program regularly works with giant lists of email data received from the inbox, often needing to be sorted by certain criteria (date, subject, etc.) as per the user's needs. As such, the program requires a robust and efficient algorithm that can perform well even when applied to larger sets of data.

Search queries and filters are also crucial for the application's organization and user ease of use. A large volume of data, even if sorted by a criteria, can often be hard to navigate through. By narrowing down what is displayed even further and more specific, the user can find the relevant information even quicker.

The email classifier is crucial for the application and the crux of its concept. Often, the topic and importance of any given email received cannot always be determined unless manually opened, read, and evaluated by a human user. Given the number of emails inside, and the number of emails a person continues to receive inside their electronic inbox every day, it can be nearly impossible to productively read through one's email inbox. But with the application of the classifier, the application seeks to automatically make the overall idea and importance of each new piece of email clear to see for the viewer from the beginning.

Chapter 3:

Conclusion and Recommendations

In conclusion, it is necessary for an email client to handle and sort large amounts of data. Along with the classifier, algorithms are a key and indispensable portion of managing, organizing, and labeling the datasets the application both receives and wishes to display to the user in a productive manner.

SVC models are strictly defined by the classes you set for it and the training data provided. On this basis, future development could experiment with other dataset and corresponding labels to expand both the versatility and applicability of the application to a wider range of users and use cases.