

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | | Email Auto Reply System | | |  | | |
|  | |  | | | | |  | |
|  |  | | | | | | |  |
|  | |  |  | | --- | --- | | **Team Members** | **IDs** | | Rawan Asaad | 20213882 | | Farah Ahmed | 20213785 | | Basant Medhat | 20213771 | | | | | | | |  |
|  | —Natural language processingSp25 - CS443—Supervised by: Eng/Farah Ashraf  Eng/Ahmed | | | | | | |  |
|  | | | |  |  | | | |

* **Approach Taken:**
* In this project, we aimed to build an automated email reply system using Natural Language Processing (NLP) techniques.
* Our objective was to classify incoming customer emails into predefined responses that address common support queries.
* We used three different models to evaluate and compare their performance:
  + 1. **Support Vector Machine (SVM):** with TF-IDF vectorization.
    2. **Logistic Regression (LR):** a simple linear model with TF-IDF features.
    3. **Recurrent Neural Network (RNN):** using an embedding layer and Simple RNN.
    4. **Long Short-Term Memory (LSTM):** with Bidirectional layers and dropout regularization.
    5. **Gated Recurrent Unit (GRU)**: a more efficient alternative to LSTM with similar capabilities.

The main steps were:

1. **Data Preparation**: Preprocessing the email text, tokenization, and label encoding.
2. **Model Training**: Training all three models on the same labeled dataset.
3. **Evaluation**: Comparing accuracy and classification metrics.

a. **Dataset**:  
We used a labeled dataset of customer support emails (customer\_email) and their corresponding replies (support\_reply).

* 1. **Text Cleaning**:  
     We applied extensive text preprocessing including:
     1. Lowercasing
     2. Removing punctuation and special characters
     3. Removing common polite suffixes like "thanks", "please", from support replies to normalize outputs.
  2. **Label Encoding**:  
     Each unique support reply was encoded into a numeric label using LabelEncoder.
  3. **Training and Testing**:
     1. The data was split into training (80%) and testing (20%) sets.
     2. For neural models, tokenization and padding were applied to convert text to sequences.
* **Results Obtained:**

|  |  |
| --- | --- |
| Model | Accuracy |
| SVM | 95.14% |
| RNN | 95.14% |
| LSTM | 95.14% |
| Logistic Regression | 94.72% |
| GRU | 95.14% |

* **Performance Metrics:**

1. **SVM:**

* **Strengths**: Accurately predicted the main version of each response.
* **Weakness**: Completely failed to detect the "if possible" variants — precision, recall, and F1-score were all **0.00** for these.
* **Macro F1-score**: **0.49**
* **Weighted F1-score**: **0.93**

1. **Logistic Regression:**

* **Strengths:** Lightweight, easy to implement.
* **Weakness:** Slightly lower accuracy; struggles with longer dependencies.
* **Macro F1-score: 0.46**
* **Weighted F1-score: 0.91**

1. **RNN:**

* **Strengths**: Accurately predicted **all** classes including the "if possible" variants. Balanced results across the board.
* **Macro F1-score**: **0.98**
* **Weighted F1-score**: **0.98**
* **Observation**: Best overall balance between accuracy and handling class imbalance.

1. **LSTM:**

* **Strengths**: Like SVM, handled main classes well.
* **Weakness**: Like SVM, struggled with rare classes — "if possible" variants had zero predicted samples.
* **Macro F1-score**: 0.49
* **Weighted F1-score**: 0.93

1. **GRU:**

* **Strengths**: Comparable to LSTM in accuracy, faster training.
* **Weakness:** Slight drop in recall for minor classes.
* **Macro F1-score: 0.50**
* **Weighted F1-score: 0.94**
* **Potential Improvements:**
* **Class Imbalance Handling**: The major issue was that models like SVM and LSTM didn't predict any samples for rare classes (e.g., “if possible” versions). Techniques such as **SMOTE**, **class weighting**, or **data augmentation** could help balance class representation.
* **Model Ensemble**: Combining SVM and RNN predictions in an ensemble could enhance generalization and accuracy.
* **Fine-Tuning with Pre-trained Embeddings**: Incorporating **BERT** or other transformer-based embeddings could further improve performance, especially on minority classes.

|  |  |  |
| --- | --- | --- |
| Technique | Targets | How It Helps |
| SMOTE / Class Weights | Class imbalance | Improves recall/F1 on rare classes |
| Ensembling | General model robustness | Balances strengths of multiple models |
| Pre-trained Embeddings (BERT) | Semantic understanding | Captures deeper context; boosts rare-class performance |