

Email Classification

Objective

This web app leverages Machine Learning to classify support emails into categories like Incident, Request, Problem, and Change. It ensures privacy by masking sensitive information, such as names, phone numbers, and email addresses, before processing. Built with Flask, the app improves support team efficiency and helps maintain data security by automating email categorization and safeguarding personal data throughout the process.

Packages used

Frontend: HTML + CSS

Backend: Flask (Python)

Model: Naive Bayes

NLP: NLTK for text cleaning & stemming

PII Masking: Regex-based masking

Deployment: Flask local server (easily deployable on cloud platforms)

Step 1: Retrieving English Language Emails

To ensure the model is trained and evaluated only on relevant and understandable content, the raw dataset is filtered to include only emails written in English. This process uses a language detection library such as langdetect, which analyzes each email's content and identifies its language. Emails not classified as English are excluded from further processing. This step enhances the model's performance and reduces noise in classification tasks.

Step 2: Masking Process

To mask Personally Identifiable Information (PII) in emails, we use regular expressions (regex) to identify and replace specific patterns of sensitive data, such as email addresses, phone numbers, credit card details, and personal identifiers (e.g., Aadhar numbers, names)

Step 3:Text Cleaning

In this step, we used the Natural Language Toolkit (NLTK) to clean and preprocess text data. The goal is to remove noise, normalize the text, and prepare it for further processing, such as classification or analysis. We'll cover text cleaning (removing unwanted characters) and stemming (reducing words to their root form) using NLTK.

Step 4: Data Transformation

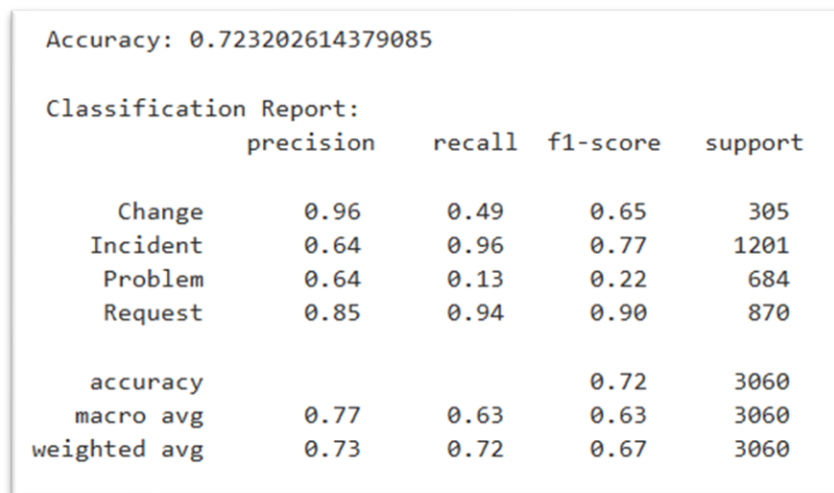
To build an effective machine learning model for email classification, it's essential to convert raw email text into a numerical format that algorithms can understand. This transformation is achieved using TF-IDF vectorization

Step 5: Model Building

After transforming the raw email text into structured TF-IDF vectors, the next step is to build a machine learning model that can learn from this data and classify new emails like as Incident, Request, Problem, and Change

For this task, I used the Multinomial Naive Bayes (MNB) algorithm — a robust and efficient method for text classification problems.

Step 6: Model Evaluation



```
Accuracy: 0.723202614379085

Classification Report:
              precision    recall  f1-score   support

   Change       0.96       0.49       0.65       305
  Incident       0.64       0.96       0.77      1201
   Problem       0.64       0.13       0.22       684
   Request       0.85       0.94       0.90       870

 accuracy              0.72       3060
 macro avg           0.77       0.63       0.63       3060
weighted avg           0.73       0.72       0.67       3060
```

Overall Accuracy: 72.32%

This means the model correctly predicted the email category for approximately 72 out of every 100 emails.

Change:

High precision (0.96) but low recall (0.49).

The model is cautious: when it predicts "Change", it's almost always correct — but it often fails to detect all the actual "Change" emails.

May be due to underrepresentation or overlap with other categories.

Incident:

High recall (0.96), meaning most "Incident" emails are correctly detected.

Precision is moderate (0.64), indicating some false positives.

The model may favor this class due to its large support.

Problem:

Both recall and F1-score are low (0.13 and 0.22), showing poor performance on this class. This category might be confused with "Incident" or "Request". You may need more representative data or better feature engineering.

Request:

Strong across all metrics. F1-score of 0.90 shows the model is very good at identifying "Request" emails.

Likely has distinct keywords and good training examples.

Step 7:Model Deployment

- **Email Classification**

- └─ **app.py**

- | Flask application file that defines routes and connects the UI with the model for predictions.

- |

- └─ **model/**

- └─ **email_classifier_model.pkl**

- | Serialized Multinomial Naive Bayes model trained on TF-IDF features.

- └─ **tfidf_vectorizer.pkl**

- | Serialized TF-IDF vectorizer used for transforming input emails into numerical features.

- |

- └─ **templates/**

- └─ **index.html**

- | HTML template for the web interface that takes email input and displays predictions.

- |

- └─ **static/**

- └─ **style.css**

- | CSS stylesheet used to style the front-end of the web app.

- |

- └─ **notebook/**

- └─ **MLmodel.ipynb**

- | Jupyter Notebook used for training and evaluating the model before deployment.

- |

- └─ **requirements.txt**

- | Lists the Python dependencies needed to run the application (Flask, scikit-learn).

Step 8: Model Final Input/Output

input

Email Classifier

Subject: Application not loading

Hi Team,My name is Rakesh Kumar. I've been trying to access the CRM system, but it keeps showing a blank screen. I tried restarting my browser and system, but the issue persists. Please look into this urgently. You can reach me at rakesh.kumar23@example.com or +91 9876543210. Thanks.

Classify

Output

Masked Email:

Subject: Application not loading [full_name]. My name is [full_name]. I've been trying to access the CRM system, but it keeps showing a blank screen. I tried restarting my browser and system, but the issue persists. Please look into this urgently. You can reach me at [email] or +[dob] [phone_number].

Preprocessed Email:

subject applic load name tri access crm system keep show blank screen tri restart browser system issu persist pleas look urgent reach email dob

Email Class:

Incident

Step 9: Challenges Faced:

PII Masking Without LLMs

Crafting regex patterns to identify and mask personal identifiable information such as emails, phone numbers, and names accurately without false positives.

Language Detection Accuracy

Filtering out non-English emails using lightweight language detection tools like langdetect, which may misclassify short or code-mixed emails.

Imbalanced Data

Some classes (e.g., "Problem") had significantly fewer samples or lower predictive accuracy, affecting model performance.

Model Overfitting

Risk of overfitting during training due to noisy data or overly complex feature spaces from TF-IDF.

Deployment & Integration

Ensuring the model runs smoothly in a Flask app and interacts properly with HTML/CSS frontend and external tools.

Step 10: Proposed Solutions:

Robust Regex-Based Masking

Use finely tuned regular expressions for each type of PII, and perform unit tests to verify masking effectiveness.

Minimum Length Thresholds for Language Detection

Ignore emails below a certain character length to improve the reliability of language filtering.

Class Balancing Techniques

Use sampling methods (e.g., SMOTE or undersampling), or apply class weights in the model to address imbalance.

TF-IDF Optimization

Limit n-gram range and max features in the TF-IDF vectorizer to reduce overfitting.

Containerized Deployment

Use Docker to deploy the Flask app consistently across different environments, and consider FastAPI for performance upgrades.