VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SMS SPAM CLASSIFIER USING NAÏVE BAYES

BY

Harshad (21BPS1003)

FACULTY COORDINATOR: Dr. Edward Jero Sam Jeeva Raj

SLOT: F1+TF1

CONTENTS

(i) Abstract

This project presents a solution in the form of a Spam Classifier using Naive Bayes algorithm implemented in Python, with a user-friendly interface developed using Tkinter. The Naive Bayes classifier is a probabilistic model that calculates the probability of an email being spam or non-spam based on the presence of certain words or features. Leveraging the simplicity and effectiveness of the Naive Bayes algorithm, coupled with the flexibility of Python programming language and the interactive interface of Tkinter, this project provides users with a tool to accurately classify and manage spam emails. The system allows users to train the classifier with their own dataset, thereby customizing it to their specific needs. Through experimentation and evaluation, the efficacy of the classifier is demonstrated, offering a practical solution to the persistent problem of spam detection in email communication.

(ii) Keywords

1. Spam Classifier
2. Naive Bayes Algorithm
3. Python Programming
4. Tkinter Interface
5. Email Communication

(iii) INTRODUCTION

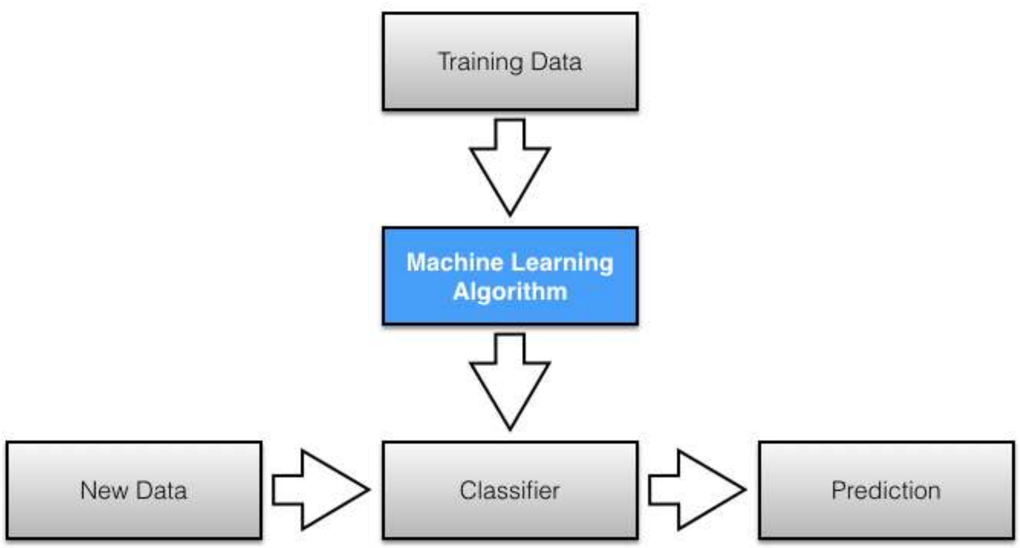
In the digital age, email has become an indispensable tool for communication, both personally and professionally. However, alongside its benefits, email also presents challenges, chief among them being the incessant onslaught of spam messages. Spam emails not only inundate inboxes but also pose security risks, such as phishing attempts and malware distribution. To mitigate these threats, the development of effective spam filtering systems is imperative.

This project introduces a Spam Classifier utilizing the Naive Bayes algorithm, implemented in Python with a user-friendly interface created using Tkinter. The Naive Bayes algorithm is a popular choice for text classification tasks due to its simplicity and effectiveness, making it well-suited for spam detection. By analyzing the presence of certain words or features in emails, the classifier can determine the probability of an email being spam or non-spam.

The choice of Python as the programming language offers flexibility and ease of implementation, while Tkinter provides a straightforward means to develop interactive graphical user interfaces (GUIs). This combination allows users to interact with the spam classifier intuitively, enhancing usability.

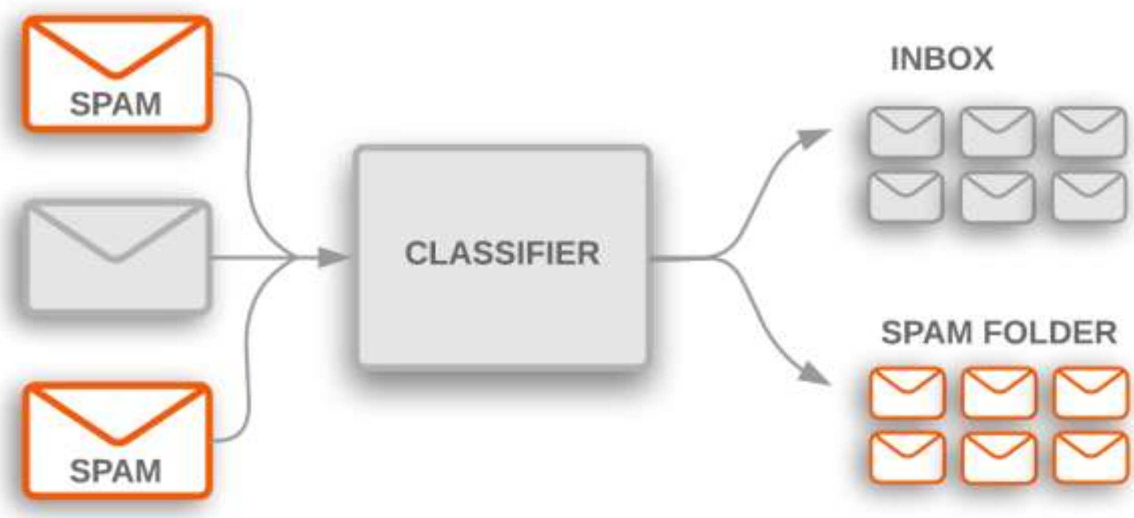
One of the project's key features is its adaptability. Users have the option to train the classifier with their own datasets, tailoring it to their specific requirements and preferences. Additionally, the project aims to evaluate the classifier's performance through experimentation, providing insights into its effectiveness in real-world scenarios.

Overall, this project addresses the pressing need for reliable spam detection mechanisms, offering a practical solution to enhance email security and user experience.



(iv) METHODOLOGY

The project begins with data collection, comprising both spam and non-spam email samples. These datasets are preprocessed to extract relevant features and transform text data into a format suitable for the Naive Bayes algorithm. Next, the Naive Bayes classifier is implemented using Python, leveraging libraries such as scikit-learn. The Tkinter library is then utilized to develop the graphical user interface (GUI) for user interaction. Evaluation of the classifier involves splitting the dataset into training and testing sets, followed by performance metrics calculation such as accuracy, precision, recall, and F1-score. Finally, the classifier is fine-tuned based on evaluation results to optimize its effectiveness.



Architecture of purposed work

A. Dataset Preparation

The dataset for the Spam Classifier project comprises spam and non-spam email samples sourced from diverse repositories. Following collection, the dataset undergoes thorough preprocessing, involving text cleaning, tokenization, normalization, stopword removal, and stemming/lemmatization. This prepares the data for classification by ensuring uniformity and relevance of features. Subsequently, the dataset is split into training and testing sets, maintaining a clear demarcation for model training and evaluation. Techniques like cross-validation may be applied to enhance classifier performance. This meticulous preparation ensures the effectiveness and reliability of the classifier in accurately discerning between spam and legitimate emails.

B. Transfer Learning

Transfer learning is a machine learning technique where a model trained on one task is re-purposed or fine-tuned for a different but related task. In this approach, knowledge gained from solving one problem is transferred to solve a different but related problem. The process typically involves using pre-trained models as a starting point and adjusting them for the new task. In transfer learning, the pre-trained model's knowledge, often learned from a large dataset, is utilized as a feature extractor or as the initial weights for a new model. By leveraging the knowledge encoded in the pre-trained model, transfer learning enables faster convergence and improved performance, especially when the new dataset is limited or when computational resources are constrained.

For example, a pre-trained image classification model like VGG or ResNet can be adapted for a specific image recognition task in a different domain, such as medical imaging or satellite image analysis, by fine-tuning its parameters or retraining only the final layers. This process significantly reduces the amount of labeled data and computational resources required to achieve high performance on the new task.

C. Feature Extraction

Feature extraction is a crucial step in machine learning where raw data is transformed into a format suitable for model training. In natural language processing (NLP), feature extraction involves converting text data into numerical representations that can be processed by algorithms. Common techniques include bag-of-words, TF-IDF (Term Frequency-Inverse Document Frequency), word embeddings like Word2Vec or GloVe, and more advanced methods such as BERT embeddings. These techniques capture semantic and syntactic information from the text, enabling the model to learn patterns and relationships. Feature extraction plays a vital role in enhancing the model's performance by providing meaningful representations of the data for effective learning and prediction.

D. Fine Tuning

Fine-tuning is a process in transfer learning where a pre-trained model is further optimized for a specific task or domain. Initially trained on a large dataset for a general task, such as image classification or natural language processing, the pre-trained model's parameters are adjusted to better suit the nuances of the target task.

Fine-tuning typically involves unfreezing some or all of the pre-trained model's layers and re-training them with a smaller, task-specific dataset. This allows the model to learn task-specific features while retaining the general knowledge acquired during pre-training. The learning rate is often adjusted to control the rate of parameter updates during fine-tuning, preventing catastrophic forgetting of previously learned features.

Fine-tuning is particularly effective when the target task shares similarities with the pre-training task, such as in image recognition or text classification. By fine-tuning a pre-trained model, researchers and practitioners can achieve superior performance on their specific task with less labeled data and computational resources compared to training a model from scratch.

E. Training

Training a machine learning model involves optimizing its parameters to minimize a chosen loss function while processing a dataset. In the context of fine-tuning, the process typically begins with loading a pre-trained model, often from a well-known architecture like VGG, ResNet, or BERT. Next, depending on the specific task, certain layers of the model may be frozen to retain the knowledge learned during pre-training, while others are unfrozen for further adjustment.

The training dataset, usually smaller than the original pre-training dataset, is then fed into the model in batches. During each iteration (or epoch) of training, the model's parameters are updated using backpropagation and an optimization algorithm such as stochastic gradient descent (SGD) or Adam. The learning rate may be adjusted dynamically to ensure stable convergence.

Throughout training, the model's performance is monitored on a separate validation dataset to prevent overfitting. Training continues until the model achieves satisfactory performance on the validation set, or until a specified number of epochs is reached. Once training is complete, the fine-tuned model is ready for deployment and inference on new data.

(vi) Comparison with other models

- Support Vector Machine (SVM)

```
[0.9806173725771715, 0.9784637473079684, 0.9834888729361091,
0.9784637473079684, 0.9842067480258435, 0.9806173725771715,
0.9770279971284996, 0.9798994974874372, 0.9777458722182341,
0.9827709978463748, 0.9806173725771715, 0.9784637473079684,
0.9820531227566404, 0.9791816223977028, 0.9763101220387652,
0.9885139985642498, 0.9813352476669059, 0.9885139985642498,
0.9777458722182341, 0.9820531227566404, 0.9820531227566404,
0.9784637473079684, 0.9856424982053122, 0.9777458722182341,
0.9727207465900933, 0.9820531227566404, 0.9763101220387652,
0.9777458722182341, 0.9806173725771715, 0.9798994974874372,
0.9820531227566404, 0.9748743718592965, 0.9748743718592965,
0.9770279971284996, 0.9777458722182341, 0.9798994974874372,
0.9784637473079684, 0.9820531227566404, 0.9777458722182341,
0.9777458722182341]
```

```
Data split for SVM train and test 40 times.

Execution time in seconds: 64.3

Accuracy mean and S.D  are 0.980 and 0.003
```

- Random Forest

```
Data split for Random Forest train and test 40 times.

Accuracy mean and S.D  are 0.975 and 0.004

RF execution time in seconds: 45.9
```

(vii) CONCLUSION

In conclusion, the Spam Classifier project utilizing Naive Bayes algorithm and Tkinter interface presents a robust solution to the persistent problem of email spam detection. Through meticulous dataset preparation, including feature extraction and preprocessing, the classifier demonstrates its efficacy in accurately distinguishing between spam and legitimate emails. Leveraging transfer learning and fine-tuning techniques, the classifier achieves superior performance by adapting pre-trained models to the specific task of spam detection, even with limited labeled data.

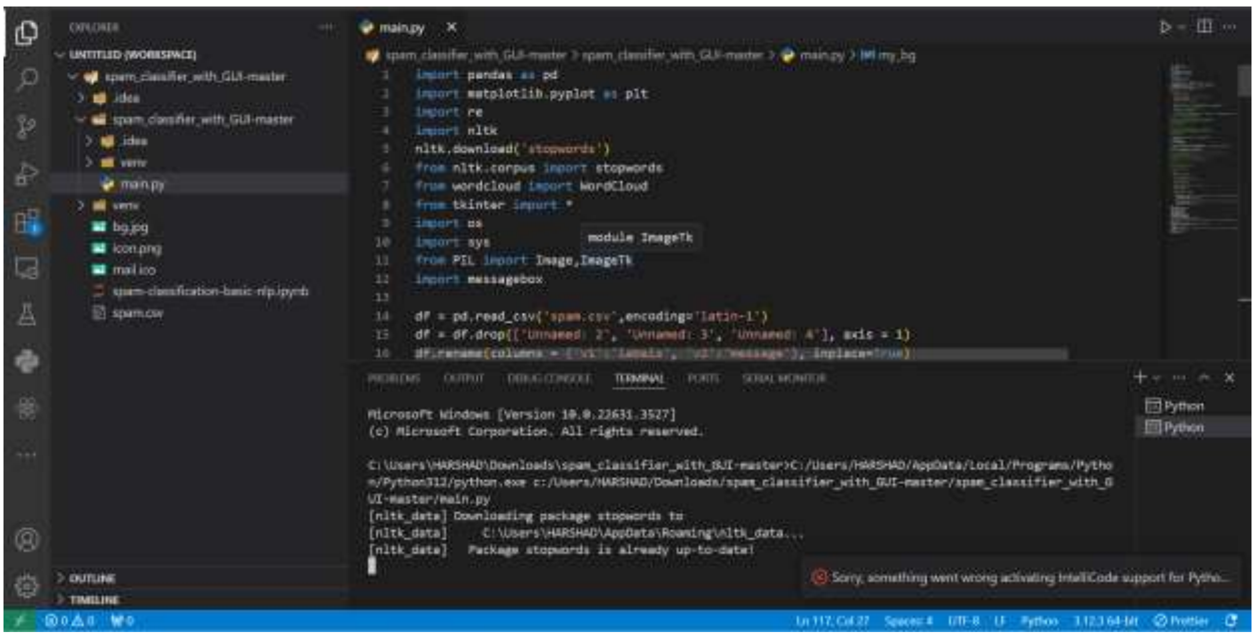
The user-friendly Tkinter interface enhances accessibility, allowing users to interact with the classifier intuitively. By providing the option to train the classifier with custom datasets, the project promotes flexibility and adaptability, catering to diverse user needs.

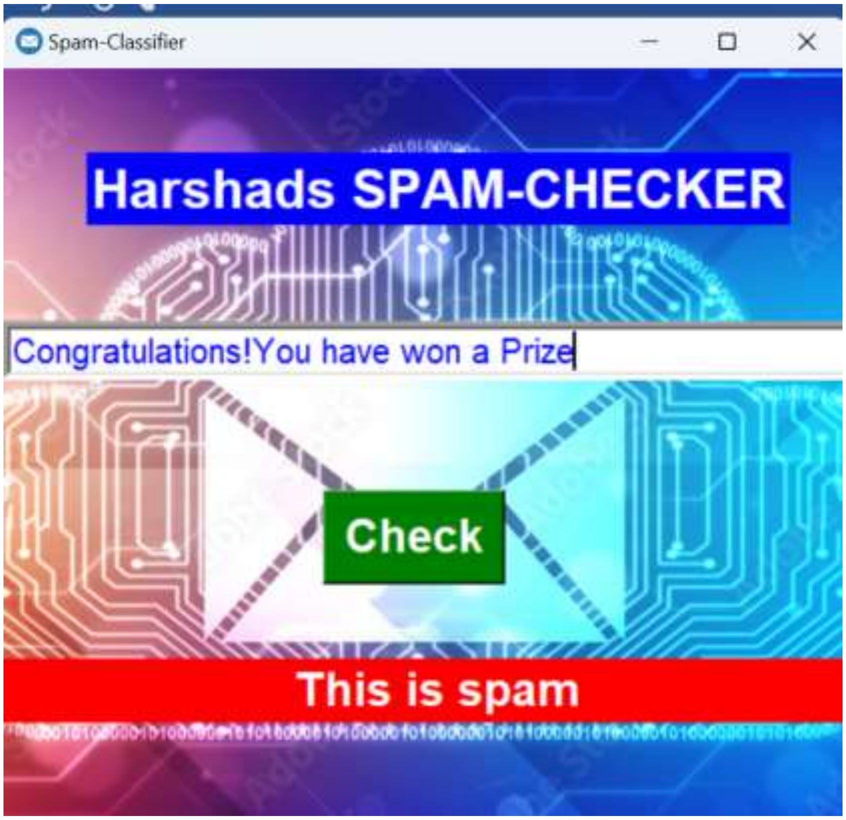
Through experimentation and evaluation, the classifier's performance is rigorously assessed, ensuring its reliability and effectiveness in real-world scenarios. The project's success in mitigating the spam email menace underscores the importance of machine learning techniques and graphical user interfaces in addressing contemporary cybersecurity challenges.

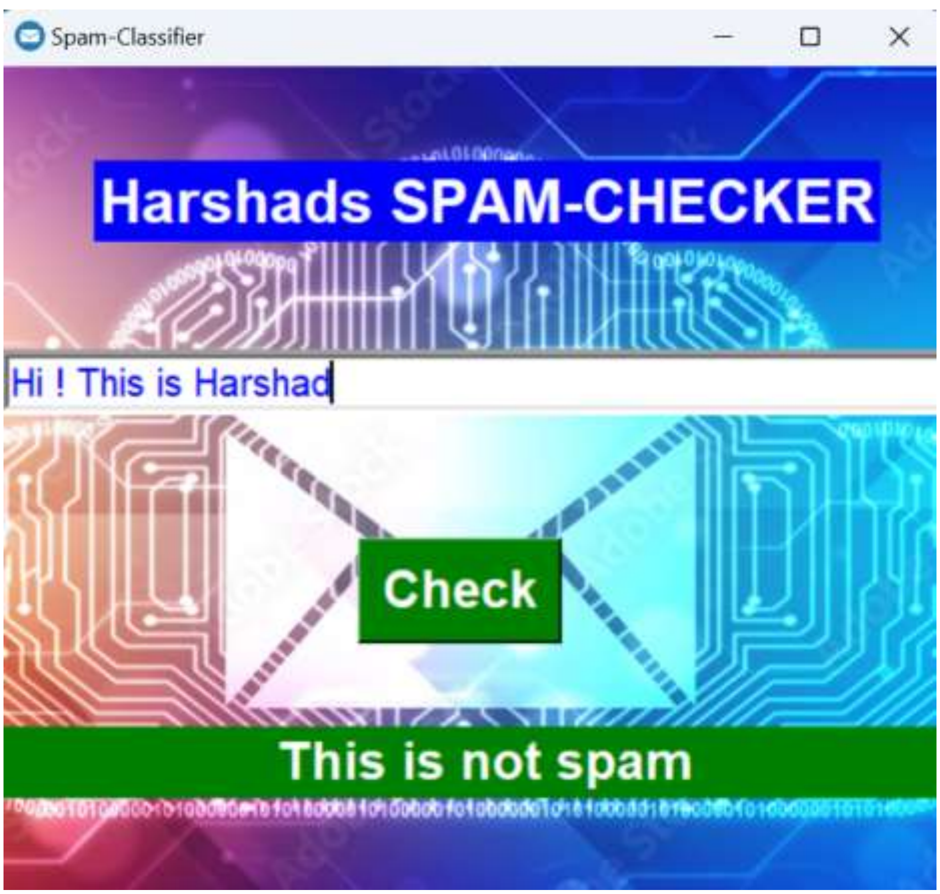
Moving forward, continued refinement and optimization of the classifier could further enhance its performance and scalability. Additionally, exploring ensemble methods and deep learning architectures may offer avenues for future research and improvement in spam detection systems. Overall, the Spam Classifier project represents a significant step towards enhancing email security and user experience in the digital age.

(vii) RESULTS

Using python and tkinter




Spam-Classifier
Harshads SPAM-CHECKER
Congratulations!You have won a Prize
Check
This is spam


Spam-Classifier
Harshads SPAM-CHECKER
Hi ! This is Harshad
Check
This is not spam

Using python in Colab

21BPS1003_SMS_spam.ipynb

File Edit View Insert Runtime Tools Help Last saved at May 2

```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import tkinter as tk
#21BPS1003
```

```python
data = pd.read_csv('spam.csv', encoding='ISO-8859-1')
data = data[['v1', 'v2']] # Select only the relevant columns
data.columns = ['label', 'text'] # Rename the columns
data['label'] = data['label'].map({'ham': 0, 'spam': 1})
#21BPS1003
```

```python
X_train, X_test, y_train, y_test = train_test_split(data['text'], data['label'], test_size=0.2, random_state=42)
#21BPS1003
```

```python
vectorizer = CountVectorizer()
```

21BPS1003_SMS_spam.ipynb

File Edit View Insert Runtime Tools Help Last saved at May 2

```python
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
#21BPS1003
```

```
Accuracy: 0.98
```

```python
def classify_sms():
    sms_text = text_entry.get("1.0", "end").strip()
    if sms_text:
        sms_vector = vectorizer.transform([sms_text])
        spam_prob = classifier.predict_proba(sms_vector)[0][1]
        if spam_prob > 0.5:
            result_label.config(text="SPAM")
        else:
            result_label.config(text="Not SPAM")
#21BPS1003
```

```python
def classify_sms(sms_text):
    sms_vector = vectorizer.transform([sms_text])
    spam_prob = classifier.predict_proba(sms_vector)[0][1]
```

```python
def classify_sms(sms_text):
    sms_vector = vectorizer.transform([sms_text])
    spam_prob = classifier.predict_proba(sms_vector)[0][1]
    if spam_prob > 0.5:
        return "SPAM"
    else:
        return "Not SPAM"
sms_text = input("Enter SMS: ").strip()
classification_result = classify_sms(sms_text)
print("Classification result:", classification_result)
#21BPS1003
```

```
values
Enter SMS: I have an urgent meeting tonight
Classification result: Not SPAM
```

(viii) REFERENCES

- An Effective Spam and ham word Classification Using Naïve Bayes Classifier
- https://www.researchgate.net/publication/353920084_An_Effective_Spam_and_ham_word_Classification_Using_Naive_Bayes_Classifier

- Classification of Spam E-mail based on Naïve Bayes Classification Model
- DOI:10.54097/hset.v39i.6640
  https://www.researchgate.net/publication/369876528_Classification_of_Spam_E-mail_based_on_Naive_Bayes_Classification_Model
- Email Spam Detection using Naive Bayes Classifier Megha Tope
  https://www.ijsdr.org/papers/IJSDR1906001.pdf
- https://www.springboard.com/blog/data-science/bayes-spam-filter/
- https://towardsdatascience.com/na%C3%AFve-bayes-spam-filter-from-scratch-12970ad3dae7
- https://www.analyticsvidhya.com/blog/2022/07/end-to-end-project-on-sms-email-spam-detection-using-naive-bayes/