
BUILDING A SMARTER AI POWERED
SPAM CLASSIFIER
SPAM
KISHORE.N

About project

The project "Building a Smarter AI-Powered Spam Classifier" is an initiative aimed at developing an intelligent system capable of effectively identifying and filtering out spam messages from a stream of electronic communications, such as emails or text messages. The primary goal is to improve the accuracy and efficiency of spam detection while minimizing the likelihood of false positives and false negatives.

PHASE - 1

In this first phase, the problem statement, design thinking process, the dataset, the data preprocessing steps and extraction techniques of the project are outlined.

PROBLEM STATEMENT

- The problem is to build an AIpowered classifier that can accurately distinguish between spam and non-spam messages in emails or text messages.

- The goal is to reduce the number of false positives and false negatives while achieving a high level of accuracy

DESIGN THINKING

- **Data Collection and Labeling**:

Gather a diverse and representative dataset containing both spam and non-spam (ham) messages. Make sure the dataset is balanced and covers various types of spam.

- **Data Preprocessing**:

Clean and preprocess the text data. This may involve tasks like lowercasing, removing special characters, stemming or lemmatization, and handling stopwords.

- **Feature Extraction**:

Convert the text data into numerical format that can be fed into a machine learning model. Common techniques include:

- **Bag-of-Words (BoW)**:

Represents text as a collection of words and their frequencies.

- **TF-IDF (Term Frequency-Inverse Document Frequency)**

Weights words based on their importance in a document relative to a corpus.

- **Word Embeddings (e.g., Word2Vec, GloVe)**

Represent words as dense vectors in a continuous vector space.

- **Deep Learning-Based Embeddings (e.g., ELMo, BERT)**

Utilize pre-trained models for contextual embeddings.

- **Model Selection**:

Choose a machine learning algorithm or deep learning architecture for the classification task. Some popular choices for text classification include:

- **Naive Bayes**

Simple and efficient, often used as a baseline.

- **Support Vector Machines (SVM)**

Effective for high-dimensional data.

- **Model Evaluation**:

Use appropriate metrics like accuracy, precision, recall, F1-score, and ROC-AUC to evaluate the performance of the model on the test set. Consider the specific requirements of your application to choose the most relevant metric.

- **Iterative Improvement**:

Depending on the initial results, you may need to refine your preprocessing, try different algorithms, or experiment with more advanced techniques like ensembling or transfer learning.

- **Post-processing and Thresholding**:

Apply post-processing techniques like setting appropriate classification thresholds to fine-tune the model's performance.

- **Continuous Monitoring and Updating**:

Keep an eye on the performance of the spam classifier over time. As spam techniques evolve, you might need to retrain or update the model to maintain its effectiveness.

- **Deployment**:

Integrate the model into your application or infrastructure. This could be on a server, cloud service, or even in edge devices, depending on your specific use case.

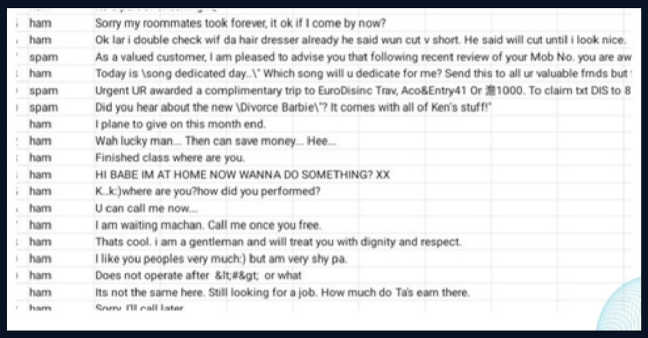
- **Feedback Loop**:

Establish a feedback loop where user feedback is used to continuously improve the model's performance.

DATASET

- The SMS spam collection is a set of SMS tagged messages that have been collected for SMS spam research.
- It contains one set of SMS messages in English of 5,574 messages, tagged according being ham(legitimate) or spam
- We can also manually label the SMS messages as either spam or not spam.This is a time consuming but crucial step

Here are some of the SPAM and HAM messages



ham	Sorry my roommates took forever, it ok if I come by now?
ham	Ok lar i double check wif da hair dresser already he said wun cut v short. He said will cut until i look nice.
spam	As a valued customer, I am pleased to advise you that following recent review of your Mob No. you are aw
ham	Today is \song dedicated day..\" Which song will u dedicate for me? Send this to all ur valuable frnds but
spam	Urgent UR awarded a complimentary trip to EuroDisinc Trav, Aco&Entry41 Or 潰1000. To claim txt DIS to 8
spam	Did you hear about the new \Divorce Barbie\"? It comes with all of Ken's stuff!"
ham	I plane to give on this month end.
ham	Wah lucky man... Then can save money... Hee...
ham	Finished class where are you.
ham	HI BABE IM AT HOME NOW WANNA DO SOMETHING? XX
ham	K..k:)where are you?how did you performed?
ham	U can call me now...
ham	I am waiting machan. Call me once you free.
ham	Thats cool. i am a gentleman and will treat you with dignity and respect.
ham	I like you peoples very much:) but am very shy pa.
ham	Does not operate after <#> or what
ham	Its not the same here. Still looking for a job. How much do Ta's earn there.
ham	Sorry, I'll call later

PHASE- 2

In this first phase, BERT is chosen as our machine learning algorithm and the steps to build the classifier is outlined

BERT- Building a spam classifier using BERT involves several steps. BERT (Bidirectional Encoder Representations from Transformers) is a powerful pre-trained language model that can be fine-tuned for various NLP tasks, including spam detection. Here's a high-level overview of the process:

1. Data Collection and Preprocessing
2. Fine-tuning BERT
3. Feature Extraction
4. Training
5. Fine-tuning Parameters
6. Evaluation
7. Testing
8. Deployment
9. Monitoring and Maintenance

1. **Data Collection and Preprocessing**:
   - Gather a dataset of labeled spam and non-spam (ham) messages.
   - Preprocess the data: remove special characters, convert to lowercase, handle numbers, and perform other necessary text cleaning steps.
2. **Fine-tuning BERT**:
   - First, you'll need to install the necessary libraries, such as TensorFlow or PyTorch, and the Hugging Face Transformers library.
   - Load the pre-trained BERT model. For instance, you can use the 'bert-base-uncased' model
3. **Data Tokenization:**
   - Tokenize your text data using the BERT tokenizer. BERT requires specific formatting of input data, including tokenization and adding special tokens for the start and end of the sequence.
4. **Train-Test Split**:
   - Split your dataset into training and testing sets.
5. **Training**:
   - Fine-tune the BERT model on your training data. You might need to adjust the learning rate, batch size, and number of epochs based on your specific dataset and requirements.
6. **Evaluation**:
   - Evaluate the model on your test data.

7. **Performance Metrics**:
    - Calculate performance metrics like accuracy, precision, recall, and F1-score to assess the model's effectiveness.
8. **Deployment**:
    - Once you're satisfied with the model's performance, you can deploy it using a suitable platform or framework. For example, you can wrap it in a Flask API or use a tool like Hugging Face's **transformers-cli** to deploy on the cloud.

STEPS TO BUILD A AI SPAM CLASSIFIER USING BERT

1. **Setting Up the Environment:**
- Install the necessary libraries: transformers, torch, numpy, and any other dependencies you might need.
2. **Load and Preprocess Data**:
- Gather a labeled dataset of spam and non-spam messages.
- Preprocess the data: remove special characters, convert to lowercase, handle numbers, and perform other necessary text cleaning steps.
3. **Fine-tuning BERT**:
- Load the pre-trained BERT model and tokenizer.
- **Tokenization and Formatting**:
- Tokenize your text data using the BERT tokenizer. BERT requires specific formatting of input data, including tokenization and adding special tokens for the start and end of the sequence.

PHASE- 3

In this third phase, the first stage of the project is developed Preprocessing and Tokenization of Text Data

1. Importing Necessary Libraries

Code:

```
import pandas as pd
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

2. Loading the Dataset

Load the dataset from a CSV file ('Spam.csv') with the specified encoding

Code:

```
data = pd.read_csv('Spam.csv', encoding='ISO-8859-1')
```

3. Removing Unnecessary Columns

Remove columns named 'Unnamed: 2', 'Unnamed: 3', and 'Unnamed: 4' from the dataset to clean it up.

Code:

```
data.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'],
axis=1, inplace=True)
```

4. Renaming Columns
Rename the remaining columns for clarity. The 'v1' column is renamed as 'label', and the 'v2' column is renamed as 'text'.

Code:

```
data.rename(columns={'v1': 'label', 'v2': 'text'}, inplace=True)
```

5. Preprocessing and Tokenization Function
Define a function to preprocess and tokenize the text data. The function performs the following steps:

Converts text to lowercase.
Removes special characters, numbers, and spaces.
Tokenizes the text.
Removes stopwords

Code:

```
def preprocess_and_tokenize(text):
 text = text.lower()
 text = re.sub(r'[^a-zA-Z\s]', '', text)
 tokens = word_tokenize(text)
 stop_words = set(stopwords.words('english'))
 tokens = [word for word in tokens if word not in stop_words]
 return tokens
```

6. Applying Preprocessing and Tokenization
Apply the preprocessing and tokenization function to the 'text' column of the dataset, creating a new column called 'tokenized_text'.

Code:

```
data['tokenized_text'] =
data['text'].apply(preprocess_and_tokenize)
```

7. Displaying the Result
Display the first few rows of the dataset, including the 'label' and 'tokenized_text' columns, to view the results of the preprocessing and tokenization.

Code:

```
print(data[['label', 'tokenized_text']].head())
```

This process involves loading a dataset, cleaning it by removing unnecessary columns, renaming columns, and then applying text preprocessing and tokenization for further analysis.

PHASE- 4

In this phase, the second stage of project is developed

Text Classification using TF-IDF and Naive Bayes

1. Importing Necessary Libraries

Code:

```
import pandas as pd
from sklearn.feature_extraction.text import
TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score,
classification_report
from sklearn.model_selection import train_test_split
```

2. Importing Preprocessed Dataset
Load a preprocessed dataset from a CSV file ('processed_spam_dataset.csv') using Pandas. The dataset should have 'text' as the feature column and 'label' as the target column.

Code:

```
data = pd.read_csv('processed_spam_dataset.csv')
X = data['text']
y = data['label']
```

3. Splitting the Data
Split the dataset into training, validation, and test sets. In this code, a 70-15-15 split is used.

Code:
```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

4. TF-IDF Vectorization
Create a TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer to convert text data into numerical feature vectors. Fit the vectorizer on the training data and transform the validation and test data.

Code:
```
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_val_tfidf = vectorizer.transform(X_val)
X_test_tfidf = vectorizer.transform(X_test)
```

5. Naive Bayes Model
Create a Multi5. Naive Bayes Model
Create a Multinomial Naive Bayes classifier and fit it to the training data.

Code:
```
model = MultinomialNB()
model.fit(X_train_tfidf, y_train)
```
nomial Naive Bayes classifier and fit it to the training data.

6. Validation Set Evaluation
Predict the labels on the validation set and evaluate the model's performance using accuracy and classification report

Code:

```
y_pred_val = model.predict(X_val_tfidf)
accuracy = accuracy_score(y_val, y_pred_val)
report = classification_report(y_val, y_pred_val)

print("Validation Accuracy:", accuracy)
print("Classification Report:\n", report)
```

7. Test Set Evaluation
Predict the labels on the test set and evaluate the model's performance using accuracy and classification report.

```
y_pred_test = model.predict(X_test_tfidf)
accuracy_test = accuracy_score(y_test, y_pred_test)
report_test = classification_report(y_test, y_pred_test)

print("Test Accuracy:", accuracy_test)
print("Test Classification Report:\n", report_test)
```

This code performs text classification using TF-IDF vectorization and a Multinomial Naive Bayes classifier, splitting the data into training, validation, and test sets, and then evaluating the model's performance on the validation and test sets.