

Abstract

Email spam classification is a critical task in today's digital world, where the amount of spam emails has increased dramatically. In this project, we propose to use machine learning (ML) and natural language processing (NLP) techniques to classify email messages as either spam or legitimate. The project aims to develop an efficient spam classifier that can accurately identify and filter spam emails from legitimate ones. The dataset used in this project will consist of a large number of email messages with their corresponding labels (spam/ham). We will use NLP techniques such as tokenization, stop word removal, stemming, and feature extraction to preprocess the text data and extract relevant features. We will evaluate several ML algorithms such as Naive Bayes, Support Vector Machines (SVMs), and Random Forests to determine the best model for spam classification. We will also perform hyper parameter tuning to optimize the model's performance. The accuracy of the classifier will be measured using evaluation metrics such as precision, recall, and F1-score. The project's outcomes will include a spam classifier model that can be integrated into an email system to automatically filter spam emails, improving email security and productivity. Additionally, the project will contribute to the advancement of NLP and ML techniques for email spam classification.

Keywords- Ham/spam, Natural Language Processing, Machine Learning, Online Platform, Email.

INTRODUCTION

Email spam has become a significant problem in today's digital age, posing challenges for individuals, businesses, and organizations alike. Spam emails are unsolicited messages that flood inboxes, wasting valuable time and resources while potentially exposing users to malicious content or scams. To combat this issue, machine learning techniques have emerged as powerful tools for email spam detection.

The objective of email spam detection is to accurately classify incoming emails as either legitimate (ham) or spam. Traditional rule-based approaches have limited effectiveness due to the constantly evolving nature of spam. Machine learning offers a more dynamic and adaptable approach by leveraging patterns and features extracted from large email datasets.

Machine learning algorithms can learn from labeled email datasets to build models capable of recognizing patterns indicative of spam. These models can then be used to automatically classify new, unseen emails. By analyzing various email attributes such as sender information, subject line, content, and embedded URLs, machine learning algorithms can identify spam characteristics and make accurate predictions.

There are several machine learning techniques commonly employed for email spam detection. These include Naive Bayes, Support Vector Machines (SVM), Decision Trees, Random Forests, and Neural Networks. These algorithms can be trained on labeled datasets, allowing them to learn the underlying patterns and relationships between spam and non-spam emails. The success of email spam detection using machine learning heavily relies on the quality and diversity of the training data. A comprehensive dataset that covers a wide range of spam types and legitimate emails is essential for training robust models. Additionally, feature engineering plays a crucial role in identifying relevant attributes and extracting meaningful information from email data. The benefits of using machine learning for email spam detection are

numerous. It enables efficient filtering and separation of legitimate emails from spam, reducing the time and effort spent by users in manually sorting through their inbox. Moreover, machine learning models can adapt to evolving spam techniques, continuously improving their accuracy over time.

In this email spam detection approach, machine learning not only enhances email security but also contributes to overall productivity and user experience. By accurately identifying and filtering spam, individuals and organizations can focus on important emails and mitigate potential risks associated with malicious content. In conclusion, email spam detection using machine learning offers a promising solution to the pervasive problem of unwanted and harmful emails. By leveraging pattern recognition and predictive models, machine learning algorithms can effectively distinguish spam from legitimate emails, enhancing email security and user experience. The continuous evolution and improvement of machine learning techniques ensure that email spam detection remains a dynamic and efficient defense against the ever-growing threat of spam.

In today's digital age, email is one of the most widely used communication mediums, and spam emails have become a significant problem for both individuals and organizations. Email spam filters are essential in managing and prioritizing emails in our inboxes. Machine learning (ML) and natural language processing (NLP) techniques can be used to develop effective email spam classifiers that can automatically identify and filter spam emails. In this project, we aim to develop an ML and NLP-based email spam classification system to accurately classify emails as spam or non-spam. The system's performance will be evaluated based on various metrics such as accuracy, precision, recall, and F1 score. The development of an accurate and efficient email spam classification system has potential to significantly improve email management and reduce the risk of fraudulent activities.

Email is one of the most popular communication methods, but unfortunately, it is also a common target for spam messages. Spam emails not only waste time but can also contain malicious links or attachments that can harm computer systems. As the volume of emails continues to grow, it has become challenging to identify and classify spam emails manually. Therefore, the development of machine learning (ML) and natural language processing (NLP) techniques has opened up new avenues for automated email spam classification. In this project, we aim to use ML and NLP techniques to classify emails as spam or legitimate, based on their content and other relevant features. The project involves building a model to analyze the text of emails and determine whether they are spam or legitimate. This study has the potential to provide a valuable solution to the problem of email spam and help users to manage their emails more effectively.

- After changing the directory, you need to open the file in below figure:

```

In [1]: import os
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import nltk
import re

from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVecorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.stem import WordNetLemmatizer
import itertools
from wordcloud import WordCloud
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

```

Reading the dataset and arranging the emails and labels into lists.

Fig.2: Changing The Directory

- Click on kernel and select restart and run all.

```

In [1]: import os
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import nltk
import re

from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVecorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.stem import WordNetLemmatizer
import itertools
from wordcloud import WordCloud
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

```

Reading the dataset and arranging the emails and labels into lists.

Fig.3: Execution Of The Project

- Wait for some time until the code gets execute, now at prediction template enter the string which you want predict whether it is a spam or ham and click on run as shown below:

Predictions

```

In [40]: em=pd.Series(['Sounds great! Are you home now? '])

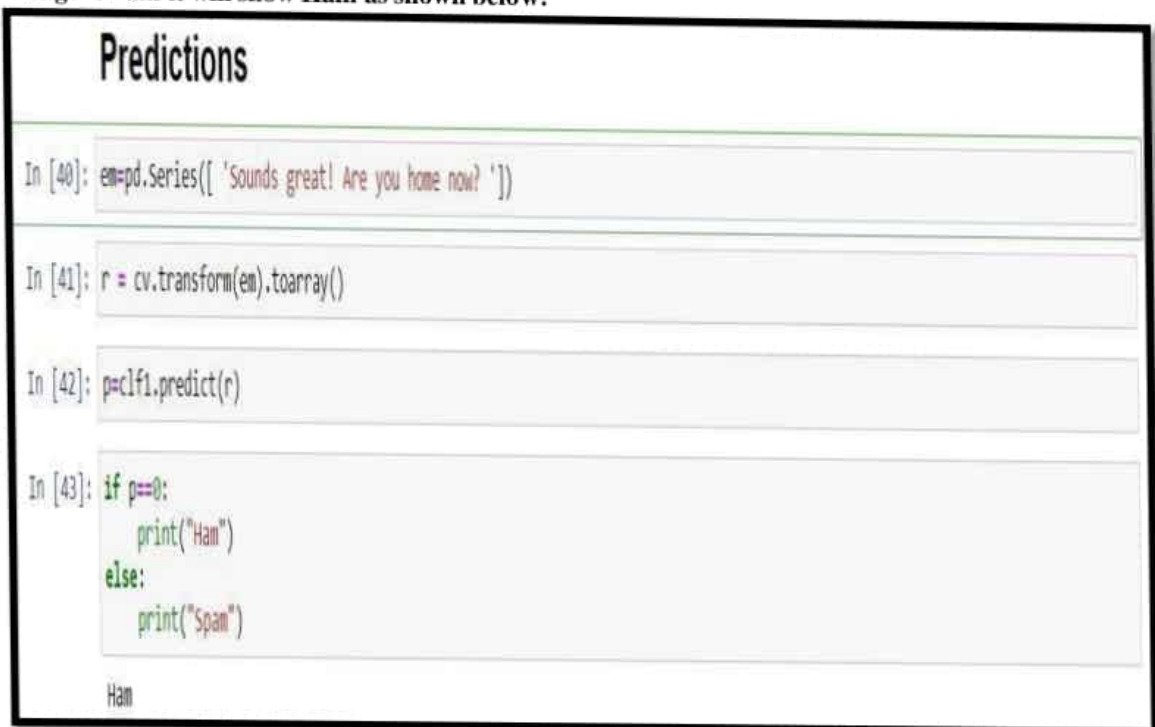
In [41]: r = cv.transform(em).toarray()

In [42]: p=clf1.predict(r)

In [43]: if p==0:
print('Ham')
else:
print('Span')

```

- If the message is ham it will show Ham as shown below:



The screenshot shows a Jupyter Notebook interface with the title "Predictions". It contains four code cells. The first cell defines a pandas Series 'em' with the text "Sounds great! Are you home now?". The second cell transforms 'em' into an array 'r' using 'cv.transform(em).toarray()'. The third cell uses a classifier 'p=clf1' to predict the class of 'r'. The fourth cell is an if-statement that prints "Ham" if the prediction is 0, otherwise it prints "Spam". Below the code cells, the output "Ham" is displayed.

```

In [40]: em=pd.Series([' Sounds great! Are you home now? '])

In [41]: r = cv.transform(em).toarray()

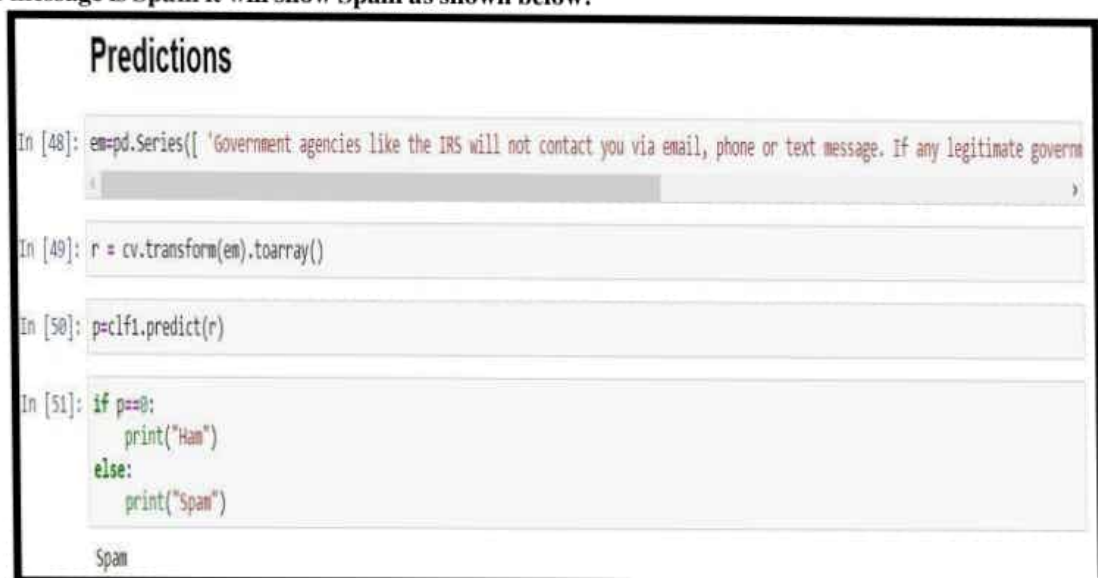
In [42]: p=clf1.predict(r)

In [43]: if p==0:
          print("Ham")
        else:
          print("Spam")

Ham
    
```

Fig.5: Prediction Of Ham

- If the message is Spam it will show Spam as shown below:



The screenshot shows a Jupyter Notebook interface with the title "Predictions". It contains four code cells. The first cell defines a pandas Series 'em' with a long text string about government agencies. The second cell transforms 'em' into an array 'r' using 'cv.transform(em).toarray()'. The third cell uses a classifier 'p=clf1' to predict the class of 'r'. The fourth cell is an if-statement that prints "Ham" if the prediction is 0, otherwise it prints "Spam". Below the code cells, the output "Spam" is displayed.

```

In [48]: em=pd.Series(['Government agencies like the IRS will not contact you via email, phone or text message. If any legitimate governa
          '
In [49]: r = cv.transform(em).toarray()

In [50]: p=clf1.predict(r)


In [51]: if p==0:
          print("Ham")
        else:
          print("Spam")

Spam
    
```

Fig.6: Prediction Of Spam

CONCLUSION

In conclusion, machine learning and natural language processing (NLP) techniques can be effectively used for email spam classification. By leveraging the power of supervised learning algorithms such as Naive Bayes, Support Vector Machines, and KNN, and by preprocessing the text data using techniques such as tokenization, stop-word removal, and stemming, it is possible to build accurate and reliable spam filters that can automatically detect and filter out unwanted emails. These techniques can also be extended to handle more complex spamming strategies such as phishing attacks and spear phishing. Overall, in the proposed models Naïve Bayes having the accuracy of 99% SVM having 98% and KNN having 97%. Finally naïve bayes having the highest accuracy so we predict the Naïve bayes model. The use of ML and NLP for email spam classification can save users valuable time and resources and improve the overall productivity and security of email communication.



```
> /*
  * Creating the Schema for our DataFrame and loading in our file
  */
import org.apache.spark.sql.types.{IntegerType, StringType, StructField,
StructType}

// Create the Schema that will represent our dataframe.
val labelField = StructField("label", StringType, nullable=true);
val emailField = StructField("emailText", StringType, nullable=true);
val fields = Seq(labelField, emailField);
val frameSchema = StructType(fields);

// Use the sqlContext to create our Dataframe
val dataCollectionDF = sqlContext
  .read
  .format("com.databricks.spark.csv")
  .option("delimited", ",")
  .option("header", "false")
  .option("mode", "PERMISSIVE")
  .schema(frameSchema)
  .load(csvFilePath);

// Cache our Dataframe so that next time it is loaded faster
dataCollectionDF.cache();
// Finally create a temp table for our data collection so we can query it
to make sure everything worked as expected
dataCollectionDF.createOrReplaceTempView("dataCollectionDF");
```

```
// Treat all urls as one generic entry
val normalizeURL = udf {
  (text: String) =>
    val regex = "(http://|https://)?www\\.\\.\\w+?\\.\\. (de|com|co.uk)"
    val pattern = Pattern.compile(regex)
    val matcher = pattern.matcher(text)

    matcher.replaceAll(" normalizedurl ")
};
```



```

> /*
  * Running our Normalization functions
  */

// We are going to run each row of the emailText column sequentially
// through each of our normalisation functions.
// This means that the first the text will be made lower case,
// then all scanned through for URLs, then all scanned
// through for email addresses etc etc.
val dataColNormalizedDF =
dataCollectionDF.withColumn("normalizedEmailText",
  removeAbnomalities(
    removeKnownNouns(
      removeShortStrings(
        removeHTMLElements(
          removeCSSElements(
            removePunctuationAndSpecialChar(
              normalizeNumber(
                removeHTMLCharacterEntities(
                  normalizeCurrencySymbol(
                    normalizeEmoticon(
                      normalizeEmailAddress(
                        normalizeURL(
                          toLowerCase(dataCollectionDF.col("emailText")))))))))))))));

// Next we cache this new table for faster loading when using it again
// later!
dataColNormalizedDF.cache();
dataColNormalizedDF.createOrReplaceTempView("dataColNormalizedDF");

```

```
// Splitting our dataCollection in to a random
// assortment of training and test data
val Array(trainingData, testData) =
dataColNormalizedDF.randomSplit(Array(0.70, 0.30))

// Defining our Variables that we will use to tweak our machine learning
algorithm
val NUM_FEATURES = 1500;
val SMOOTHING = 0.00001;
val MODEL_TYPE = "multinomial";

// Creating of Pipeline sections and defining their input and output
columns
val indexer = new StringIndexer()
    .setInputCol("label")
    .setOutputCol("indexedLabel");
val tokenizer = new Tokenizer()
    .setInputCol("normalizedEmailText")
    .setOutputCol("tokens");
val hashingTF = new HashingTF()
    .setInputCol("tokens")
    .setOutputCol("features")
    .setNumFeatures(NUM_FEATURES);

// Creating the Classifier section of our Pipeline
val naiveB = new NaiveBayes()
    .setLabelCol("indexedLabel")
    .setFeaturesCol("features")
    .setSmoothing(SMOOTHING)
    .setModelType(MODEL_TYPE);

// Actually BUILD the Pipeline
val pipeline = new Pipeline()
    .setStages(Array(indexer, tokenizer, hashingTF, naiveB));

// Then create our model by fitting the training data to our pipeline
val model = pipeline.fit(trainingData);
```




```
> // Create our Prediction from our model and testData
val prediction = model.transform(testData);

// Get totals of each email type to create accuracy percentages
val totalSpamEmails = prediction.select("*")
    .where(prediction("indexedLabel")==1.0).count().toFloat;

val totalNonSpamEmails = prediction.select("*")
    .where(prediction("indexedLabel")==0.0).count().toFloat;

val totalEmails = prediction.select("*").count().toFloat;

// Build variables for the Confusion Matrix
val truePositives = prediction.select("*")
    .where(prediction("indexedLabel")==1.0)
    .where(prediction("prediction")==1.0)
    .count().toFloat;

val falsePositives = prediction.select("*")
    .where(prediction("indexedLabel")==0.0)
    .where(prediction("prediction")==1.0)
    .count().toFloat;

val trueNegatives = prediction.select("*")
    .where(prediction("indexedLabel")==1.0)
    .where(prediction("prediction")==0.0)
    .count().toFloat;

val falseNegatives = prediction.select("*")
    .where(prediction("indexedLabel")==0.0)
    .where(prediction("prediction")==0.0)
    .count().toFloat;

// Calculate our Success at Predictions
val correctSpamPredictionRate = (truePositives / totalSpamEmails) * 100;
val correctNonSpamPredictionRate = (falsePositives / totalNonSpamEmails)
    * 100;
val correctPredictionRate = ((truePositives + trueNegatives) /
    (totalEmails)) * 100;
```