

Phishing Detection using Natural Language Processing

Installing NLTK Library for Natural Language Processing (NLP)

The NLTK is a Python package that assists developers and data scientists in dealing with large amounts of data. It can easily handle large amounts of text. NLTK can be installed by using the following Command:

\$ pip install -U nltk

```
(kali㉿kali)-[~]
$ pip install -U nltk
Defaulting to user installation because normal site-packages is not writeable
Collecting nltk
  Downloading nltk-3.9.1-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: click in /usr/lib/python3/dist-packages (from nltk) (8.1.6)
Requirement already satisfied: joblib in ./local/lib/python3.11/site-packages (from nltk) (1.4.2)
Collecting regex<=2021.8.3 (from nltk)
  Downloading regex-2024.9.11-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (40 kB)
    40.5/40.5 kB 1.7 MB/s eta 0:00:00
Requirement already satisfied: tqdm in /usr/lib/python3/dist-packages (from nltk) (4.64.1)
Downloading nltk-3.9.1-py3-none-any.whl (1.5 MB)
    1.5/1.5 MB 12.1 MB/s eta 0:00:00
Downloading regex-2024.9.11-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (792 kB)
    792.8/792.8 kB 20.4 MB/s eta 0:00:00
Installing collected packages: regex, nltk
  WARNING: The script nltk is installed in '/home/kali/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed nltk-3.9.1 regex-2024.9.11
```

Open the python terminal using the command: -

\$ python3

```
(kali㉿kali)-[~]
$ python
Python 3.11.8 (main, Feb 7 2024, 21:52:08) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Now import NLTK using the command: -

>>> import nltk

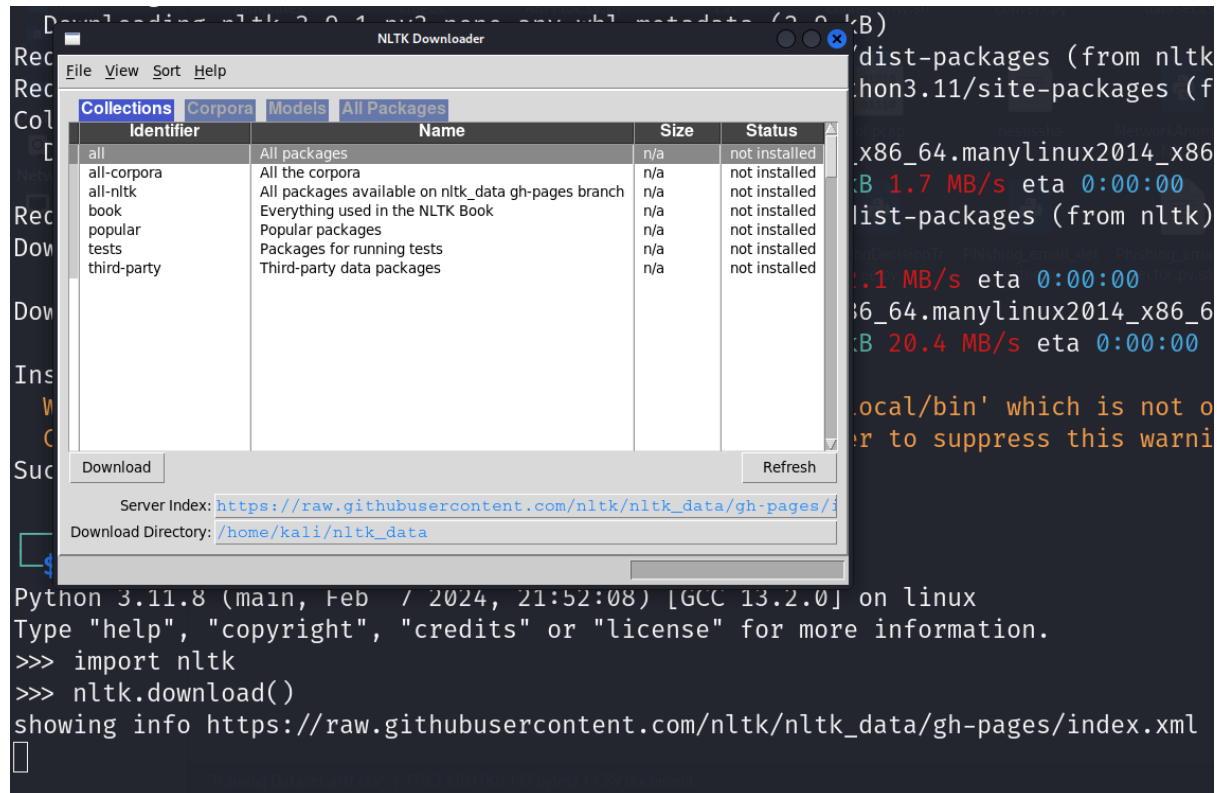
```
(kali㉿kali)-[~]
$ python
Python 3.11.8 (main, Feb 7 2024, 21:52:08) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import nltk
```

We can install the NLTK packages using: -

```
>>> nltk.download()
```

```
>>> nltk.download()
```

Entering this will display the following window: -



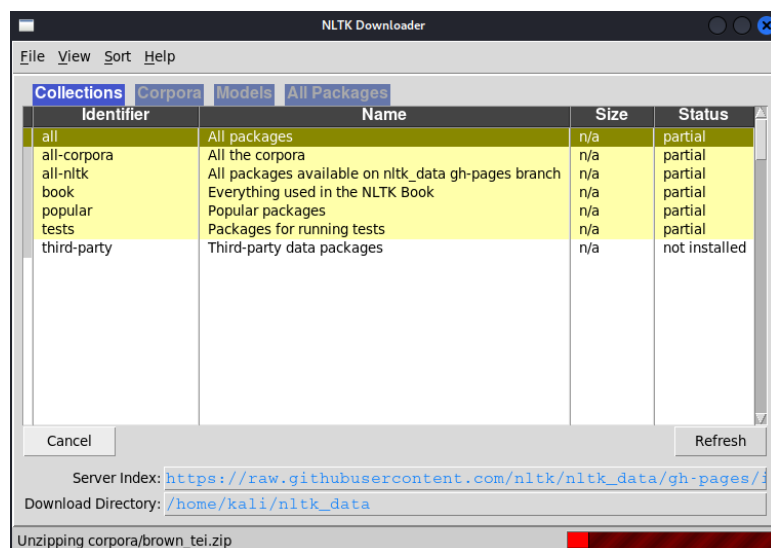
The screenshot shows the NLTK Downloader window and a terminal window. The NLTK Downloader window has tabs for Collections, Corpora, Models, and All Packages. The 'Collections' tab is active, showing a table with columns: Identifier, Name, Size, and Status. The 'all' collection is selected, and its status is 'not installed'. The 'Download' button is visible. The terminal window shows the command `nltk.download()` being executed, and the output indicates that the NLTK data is being downloaded from a GitHub repository.

Identifier	Name	Size	Status
all	All packages	n/a	not installed
all-corpora	All the corpora	n/a	not installed
all-nltk	All packages available on nltk_data gh-pages branch	n/a	not installed
book	Everything used in the NLTK Book	n/a	not installed
popular	Popular packages	n/a	not installed
tests	Packages for running tests	n/a	not installed
third-party	Third-party data packages	n/a	not installed

Server Index: https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
Download Directory: /home/kali/nltk_data

Python 3.11.8 (main, Feb / 2024, 21:52:08) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import nltk
>>> nltk.download()
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml

Select “all” and click on download. This will download all the packages in NLTK.

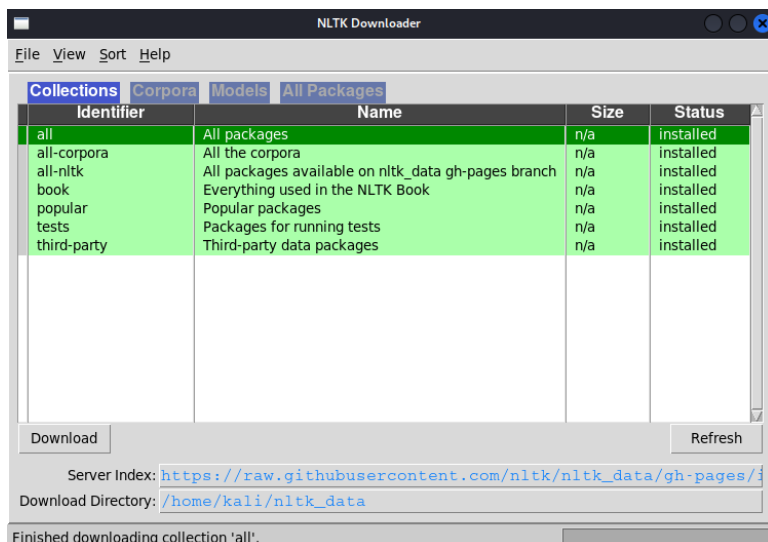


The screenshot shows the NLTK Downloader window with the 'all' collection selected. The 'Status' column now shows 'partial' for the 'all' collection, indicating that the download is in progress. The 'Download' button is still visible. The terminal window shows the command `nltk.download()` being executed, and the output indicates that the NLTK data is being downloaded from a GitHub repository.

Identifier	Name	Size	Status
all	All packages	n/a	partial
all-corpora	All the corpora	n/a	partial
all-nltk	All packages available on nltk_data gh-pages branch	n/a	partial
book	Everything used in the NLTK Book	n/a	partial
popular	Popular packages	n/a	partial
tests	Packages for running tests	n/a	partial
third-party	Third-party data packages	n/a	not installed

Server Index: https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
Download Directory: /home/kali/nltk_data

Unzipping corpora/brown_tei.zip



After all the packages are installed, close the window and press “ctrl+d” to jump out of the python terminal.

Creating New File for Writing the Code

Open a file in any editor of your choice. Here we will be using nano editor. Type in the following command: -

\$ nano file_name.py



The entire code can be obtained from: -

https://github.com/avnishnaithani/pythonforsecurity/blob/main/Domain%204/NLP_Phishing1.py

Importing the installed Libraries

In the editor, enter the following commands to import the installed libraries for running the code: -

```

kali@kali: ~
File Actions Edit View Help
GNU nano 7.2 NLP_Phishing1.py
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
import nltk
import joblib
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download('punkt')
nltk.download('stopwords')
df = pd.read_csv("spam.csv", encoding="latin-1")
df = df.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
df = df.rename(columns={"v1": "label", "v2": "text"})

```

Load the Dataset

First, download the dataset from Kaggle or from the given [link](https://github.com/avnishnaithani/pythonforsecurity/blob/main/Domain%204/spam.csv) (After you click on download, you will be prompted to create a Kaggle account if you don't have one.) or you can directly download it from this GitHub repository: -

<https://github.com/avnishnaithani/pythonforsecurity/blob/main/Domain%204/spam.csv>

Load a dataset containing labelled email data (spam or not spam). If created Python file is in the same path or directory as the downloaded dataset, using only the file name while loading dataset through `pd.read_csv()` should suffice. If they are in different path, enter the entire path along with file name (eg: `home/kali/spam.csv`) We also rename the csv header from `v1` and `v2` to `label` and `text` for better code readability.

```

df = pd.read_csv("spam.csv", encoding="latin-1")
df = df.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
df = df.rename(columns={"v1": "label", "v2": "text"})

```

Pre-processing the Dataset

In order to read the input text data efficiently, we need to preprocess the data. To achieve this, we will perform tokenization and also eliminate stop words. The text contains a lot of symbols that are

useless for our project. To get only what we need, we use tokenization. We also need to remove stop words, such as: of, is, the, and so on. This can be achieved by entering the code snippet given below: -

```
# Preprocess text data
df["text"] = df["text"].str.lower()
df["text"] = df["text"].apply(word_tokenize)
stop_words = set(stopwords.words("english"))
df["text"] = df["text"].apply(lambda x: [word for word in x if word not in stop_words])
df["text"] = df["text"].apply(lambda x: " ".join(x))
```

Feature Extraction

Next, we begin the feature extraction using Count Vectorization to convert text data into numerical features.

```
#Feature Extraction
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df["text"])
```

Split the Dataset

Next, split the dataset into training and testing data.

```
#Split the Dataset
X_train, X_test, y_train, y_test = train_test_split(X, df["label"], test_size=0.2, random_state=42)
```

Train the Classification Model

Based on the above we will train the data using a classification model, such as Multinomial Naive Bayes.

```
#Train a classification model, such as Multinomial Naive Bayes
classifier = MultinomialNB()
classifier.fit(X_train, y_train)
```

Evaluate the Model

Finally, we evaluate the performance of our model. Enter the code given below: -

```
#Evaluate the model's performance using metrics like accuracy and classification report
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(report)
```

Using the Model for Real World Data

We had split our dataset to training and testing data. Our model is trained using the training data and based on that, it reads the remaining test data and evaluate the performance by calculating the accuracy rate.

To use this model on real world data we add the following to the code: -

```
joblib.dump(classifier, 'email_spam_model.pkl')
loaded_model = joblib.load('email_spam_model.pkl')
new_email = [input("Message:")]
new_email = vectorizer.transform(new_email) # Assuming you have the vectorizer from the previous code
prediction = loaded_model.predict(new_email)

if prediction[0] == "spam":
    print("This email is spam.")
else:
    print("This email is not spam.")
```

This will ask the user for an input. Through this we can copy or enter the required text to detect if it is spam or not spam.

Save the File

To save this file press “CTRL+x”, then press “y” and finally press enter.

Convert The File to an Executable File

For this, use the command: -

Chmod +x file_name.py

```
(kali@kali)~$ chmod +x NLP_Phishing1.py
```

Run The Code

Finally Run the Python program using the command: -

Python3 file_name.py

```
(kali㉿kali)-[~]
$ python3 NLP_Phishing1.py
```

Output

```
(kali㉿kali)-[~]
$ python3 NLP_Phishing1.py
[nltk_data] Downloading package punkt to /home/kali/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /home/kali/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
Accuracy: 0.9748878923766816
Classification Report:
      precision    recall  f1-score   support

   ham       0.99       0.98       0.99       965
  spam       0.90       0.92       0.91       150

 accuracy         0.97       1115
macro avg       0.94       0.95       0.95       1115
weighted avg    0.98       0.97       0.98       1115

Message:
```

This is the part where we evaluate this model.

Ham: Refers to legitimate, non-spam messages.

Spam: Refers to unwanted, potentially malicious messages.

accuracy: The ratio of correct predictions (both true positives and true negatives) to the total number of cases examined.

Macro avg: The unweighted mean of the metric for each class. It doesn't consider class imbalance. Macro average is like treating each class equally, regardless of how many examples it has. It calculates the metric (like precision or recall) for each class separately and then takes a simple average. This approach gives equal importance to all classes.

Weighted avg: The weighted average of the metric for each class, taking into account the number of instances for each class. Weighted

average takes into account how many examples each class has. It gives more importance to classes with more examples.

Precision: The ratio of correctly predicted positive instances to the total predicted positive instances. It answers the question: "Of all the instances the model labeled as positive, how many actually were positive?"

Recall: The ratio of correctly predicted positive instances to all actual positive instances. It answers: "Of all the actual positive instances, how many did the model correctly identify?"

f1-score: The harmonic mean of precision and recall, providing a single score that balances both metrics. It's particularly useful when you have an uneven class distribution.

Support: The number of occurrences of each class in the test dataset.

The model performs slightly better on identifying ham (0.99 f1-score) than spam (0.91 f1-score).

There's a class imbalance in the dataset, with many more ham instances (965) than spam (150).

The overall accuracy of the model is 0.97 or 97%.

The weighted average metrics are slightly higher than the macro average due to the class imbalance and better performance on the majority class (ham).

We will also be prompted to enter a message whose legitimacy we want to check: -

```
Message:Hurry! Click on the link below to claim your free gift card.
This email is spam.
```

```
Message:Please find below the official mail Id. Kindly use the same for accessing the official documents.
This email is not spam.
```

```
(kali@kali)-[~]
$
```


We can now use our model to work with real world data from messages and emails to check if it is spam or not spam.