IMARTICUS LEARNING

Headline Sarcasm Classification And Article Link Validity Classification

Made By
Vikash Singh Bafila

Under Guidance Of
Prof. Ritik Raj Vaishya

Problem Statement

- In today's digital age, automated text classification has gained significant importance for managing and processing the ever-growing volume of textual data available online. This project focuses on two primary tasks: Sarcasm Detection using Natural Language Processing (NLP) and Checking the Validity of a Provided URL/Link.

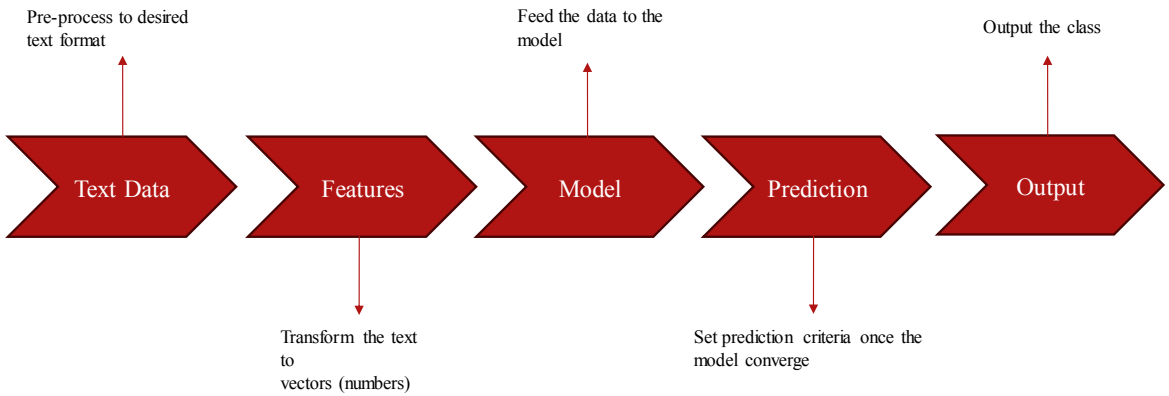
Objective

- **Sarcasm Detection using Natural Language Processing (NLP):**
  - Develop a model for detecting sarcasm in textual data, particularly headlines.
  - The program should be able to take a text as input and predict whether it is sarcastic or not.
  - Utilize NLP techniques and machine learning algorithms to build an accurate sarcasm detection model.
  - The model should classify the input text as sarcastic or non-sarcastic.

Checking the Validity of a Provided URL/Link:

- Create a function that can check the validity of a given URL or link.
- The program should ensure that the URL is accessible and does not return errors.
- It should also verify if the URL follows a standard format and is not broken or malicious.
- The function should return a result indicating whether the provided URL is valid or not.

General Process



Data Pre-Processing

IMARTICUS LEARNING

➢Open-source dataset sample

1. Dataset downloaded from kaggle

2. Contains 3 variables: article_link, headline, is_sarcastic

➢Dictionary or vocabulary which is used to train the mode

1.Either tagged (for supervised learning) or untagged (for unsupervised)

2. Size depends on the algorithm used. Should be preprocessed to remove unwanted characters, to convert to wanted format, etc.

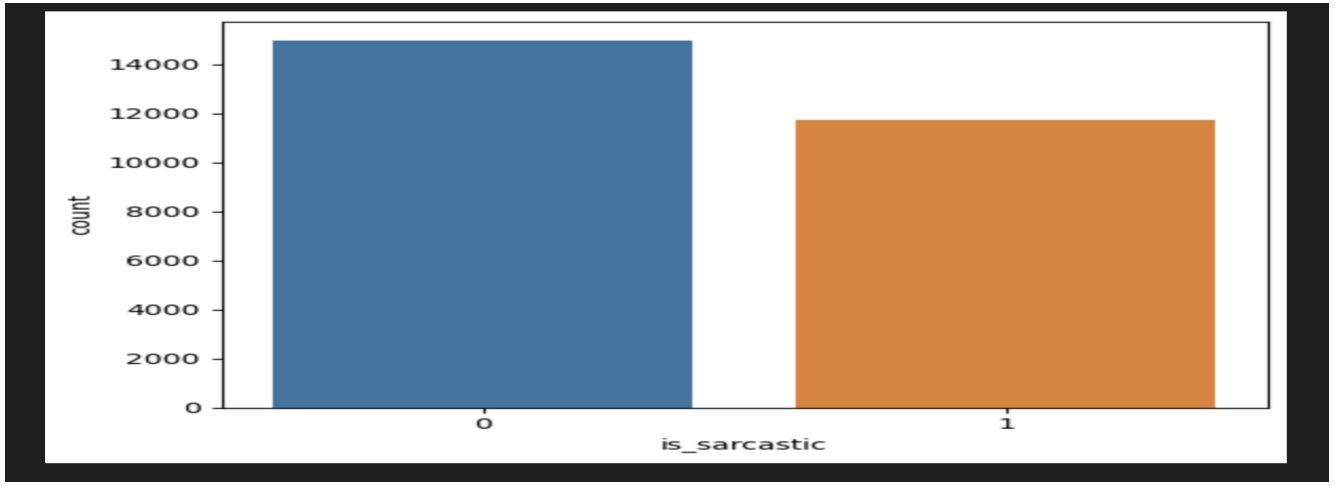
Data Pre-Processing

EDA: Finding unique values, No. Of duplicate rows and total words in all headline variable.

```
null values:

 article_link    0
headline         0
is_sarcastic     0
dtype: int64
---/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/---
Duplicate rows: 26709
After Removal 0
---/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/---
Shape of the data:

 (26708, 3)
---/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/---

We have 263022 words in the headline

---/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/------/---

is_ sarcastic value count:

 is_sarcastic
0    14984
1    11724
Name: count, dtype: int64
```

Data Pre-Processing

EDA: Displaying Count Plot



Feature Extraction

Text Data Input:

- Feature extraction typically starts with a collection of text documents, which could be sentences, paragraphs, or entire documents.

•**Tokenization**:

- The first step is to break down the text data into smaller units, often words or tokens. Tokenization segments the text into meaningful units, making it easier to analyze.

•**Vectorization Methods**:

- There are several vectorization methods commonly used for feature extraction:

a. **Count Vectorization (CountVectorizer)**:

- This method represents each document as a numerical vector. The vector's dimensions correspond to the unique words in the vocabulary.
- The values in the vector are the counts of how many times each word from the vocabulary appears in the document.
- The resulting vectors are often sparse (mostly containing zeros) since most words do not appear in a given document.

b. **TF-IDF Vectorization (TfidfVectorizer)**:

- TF-IDF represents documents based on the importance of words rather than their raw counts.
- It calculates a weight for each word in a document, which reflects its importance in that document relative to its rarity across the entire corpus.

Model Selection

IMARTICUS LEARNING

- Multinomial Naive Bayes (Multinomial NB) is a popular algorithm for text classification and is particularly well-suited for handling discrete data like word counts in text documents. It is based on the principles of the Naive Bayes algorithm, which is a probabilistic classification method.

Here's how Multinomial Naive Bayes works in text classification:

Probability Model:

Multinomial NB assumes that the features (words or tokens) in a text document are generated from a multinomial distribution. In other words, it models the probability of observing a particular word in a document.

Feature Representation:

Each document is represented as a vector of features. These features are typically the word frequencies, such as the number of times each word appears in the document.

Prior and Conditional Probabilities:

Multinomial NB estimates two types of probabilities: a. Class Prior Probability (P(C)): This represents the prior probability of a document belonging to a particular class (e.g., spam or not spam). b. Conditional Probability (P(T|C)): This represents the probability of observing a set of features (words) given the class. In other words, it is the probability of a particular word occurring in a document of a specific class.

Model Selection

IMARTICUS LEARNING

➢Bernoulli Naive Bayes (Bernoulli NB) is a variation of the Naive Bayes algorithm, which is commonly used for classification tasks, especially in natural language processing (NLP) applications. Bernoulli NB is specifically designed for binary data, where features are either present (1) or absent (0), making it suitable for text classification tasks. Here's how Bernoulli Naive Bayes works:

- **Assumption of Conditional Independence**:
  - Like other Naive Bayes variants, Bernoulli NB relies on the assumption of conditional independence among features. This means that it assumes that the presence or absence of one feature is independent of the presence or absence of other features, given the class label. In the context of text classification, features are typically words or tokens.
- **Training Phase**:
  - During the training phase, Bernoulli NB learns the probabilities of the presence or absence of each feature (word or token) for each class label. It calculates two essential probabilities for each feature:
  - a. **P(feature = 1 | class)**: This is the probability that a particular feature (word) is present in documents of a specific class. It is estimated by counting the number of documents in the class that contain the feature divided by the total number of documents in that class.
  - b. **P(feature = 0 | class)**: This is the probability that a particular feature (word) is absent in documents of a specific class. It is estimated by counting the number of documents in the class that do not contain the feature divided by the total number of documents in that class.

Model Selection

IMARTICUS LEARNING

➢Support Vector Machines (SVM) are a class of supervised machine learning algorithms that can be used for various tasks, including text classification, which could be the "above problem" you mentioned. If you've vectorized your text data and are using it for classification, here's how a Support Vector Classifier (SVC) works in the context of text classification:

- **Data Preparation**:
  - You've already preprocessed and vectorized your text data using methods like TF-IDF, Count Vectorization, or others. Each document is represented as a feature vector.
- **Defining the Problem**:
  - In text classification, you typically have a set of labeled examples, where each example is associated with a category or class. The goal is to build a model that can classify new, unlabeled documents into these categories based on their feature vectors.
- **SVC Model Training**:
  - The Support Vector Classifier (SVC) is a type of SVM specifically designed for classification tasks. It works by finding a hyperplane that best separates the data into different classes.
  - During training, the SVC learns this hyperplane based on the feature vectors and their corresponding class labels. It seeks to maximize the margin between classes while minimizing the classification error.

Model Selection

IMARTICUS LEARNING

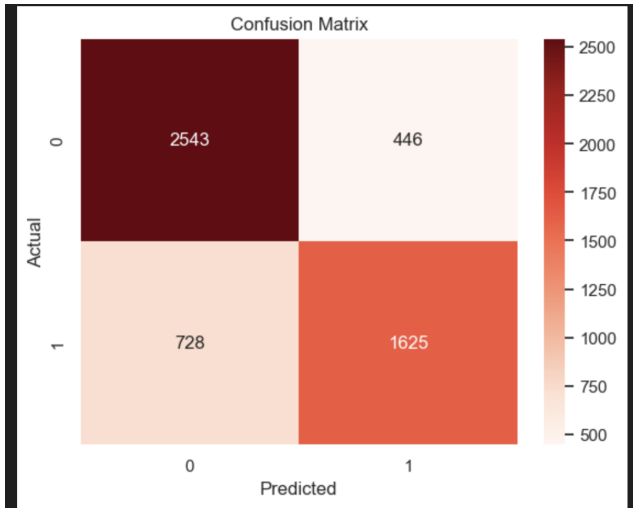
➢Support Vector Machines (SVM) are a class of supervised machine learning algorithms that can be used for various tasks, including text classification, which could be the "above problem" you mentioned. If you've vectorized your text data and are using it for classification, here's how a Support Vector Classifier (SVC) works in the context of text classification:

- **Data Preparation**:
  - You've already preprocessed and vectorized your text data using methods like TF-IDF, Count Vectorization, or others. Each document is represented as a feature vector.
- **Defining the Problem**:
  - In text classification, you typically have a set of labeled examples, where each example is associated with a category or class. The goal is to build a model that can classify new, unlabeled documents into these categories based on their feature vectors.
- **SVC Model Training**:
  - The Support Vector Classifier (SVC) is a type of SVM specifically designed for classification tasks. It works by finding a hyperplane that best separates the data into different classes.
  - During training, the SVC learns this hyperplane based on the feature vectors and their corresponding class labels. It seeks to maximize the margin between classes while minimizing the classification error.

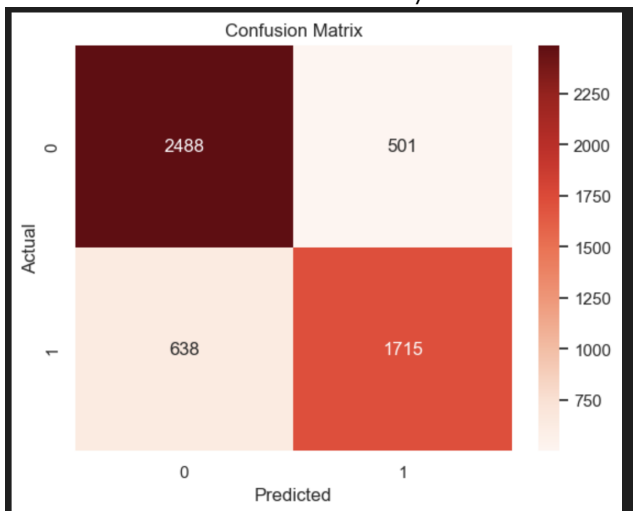
Model Evaluation

IMARTICUS LEARNING

Multinomial Navie Bayes



Bernoulli Naive Bayes



Link Validation

- Check Link Validity

1. Define a function to check the validity of a link
2. Prompt the user for a link and check its validity

```
import requests
import pandas as pd

def is_valid_link(link):     # Function to check if a link is valid
    try:
        response = requests.get(link)
        if response.status_code >= 200 and response.status_code < 300:
            return True
        else:
            return False
    except requests.exceptions.RequestException:
        return False
user_link = input("Enter a link to check its validity: ").    # User input for a link
if is_valid_link(user_link):       # Check the validity of the user-provided link
    print("The provided link is valid.")
else:
    print("The provided link is invalid.")
```

Conclusion

This project successfully combines two essential functions: sarcasm detection through NLP and URL validity checking. We implemented Naive Bayes and LSTM models for sarcasm detection, offering a versatile NLP framework. The link validity checker examines HTTP response codes to determine the accessibility of URLs. Future improvements can include advanced NLP models and comprehensive link validation methods. And a user-friendly interface can enhance accessibility. In conclusion, this project showcases the power of NLP for sentiment analysis and the importance of reliable URL validation for practical applications. It provides a solid foundation for further NLP and web utility developments.

Thank You