Web Mining Final Project Report

Helin Aylin KABAK
190709036
Hakan Karaotcu
180709050

Project Topic : A learning method to classify an arbitrary Web page as blog or not blog, for crawling purposes.

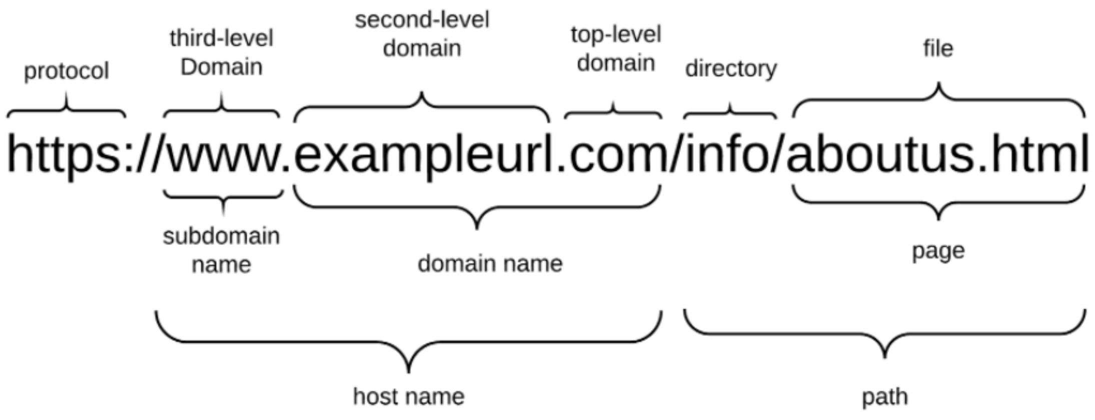
Distribution of Tasks : The tasks were completed simultaneously with online interviews.

Data Link : https://www.kaggle.com/datasets/ozgurdogan646/blog-or-not-dataset

Methods Used : Multinomial Naive Bayes (MNB)

What is MNB?

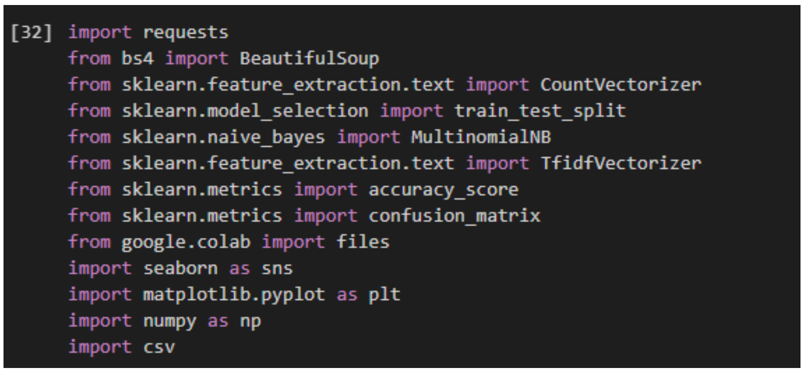
The multinomial naïve Bayes is widely used for assigning documents to classes based on the statistical analysis of their contents. It provides an alternative to the "heavy" AI-based semantic analysis and drastically simplifies textual data classification.



Basic URL Structure

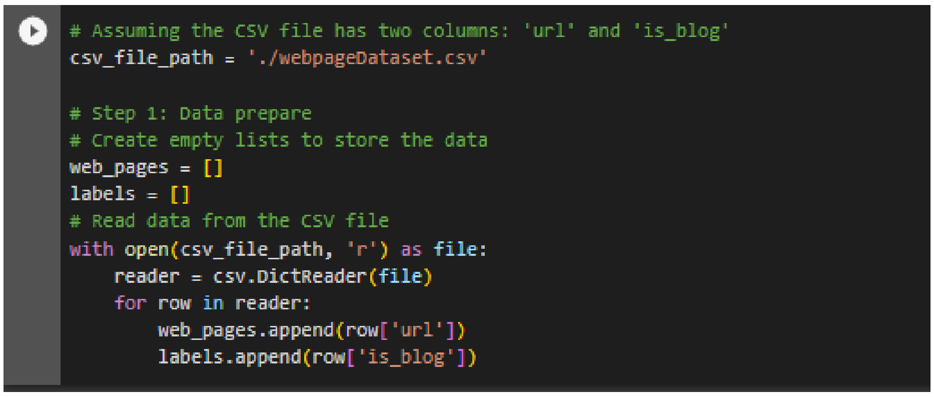
Codes:

- First of all we imported the necessary libraries.



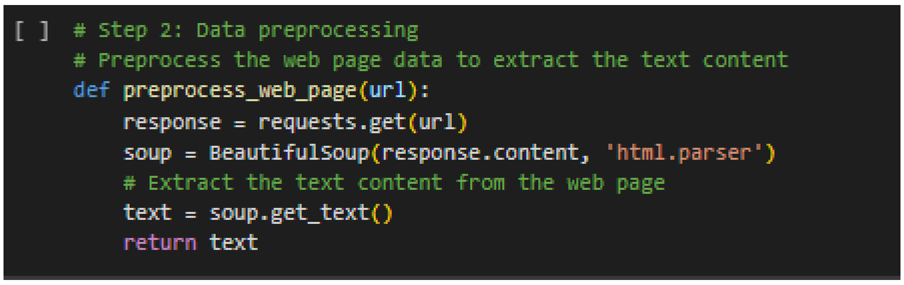
```
[32] import requests
     from bs4 import BeautifulSoup
     from sklearn.feature_extraction.text import CountVectorizer
     from sklearn.model_selection import train_test_split
     from sklearn.naive_bayes import MultinomialNB
     from sklearn.feature_extraction.text import TfidfVectorizer
     from sklearn.metrics import accuracy_score
     from sklearn.metrics import confusion_matrix
     from google.colab import files
     import seaborn as sns
     import matplotlib.pyplot as plt
     import numpy as np
     import csv
```

- This block of code opens the CSV file using the open function in read mode ('r'). It then creates a csv.DictReader object reader to read the file. The csv.DictReader treats the first row of the CSV file as the header and allows you to access the data using column names. After executing this code, the web_pages list will contain the URLs of the web pages, and the labels list will contain the corresponding labels indicating whether each web page is a blog or not.



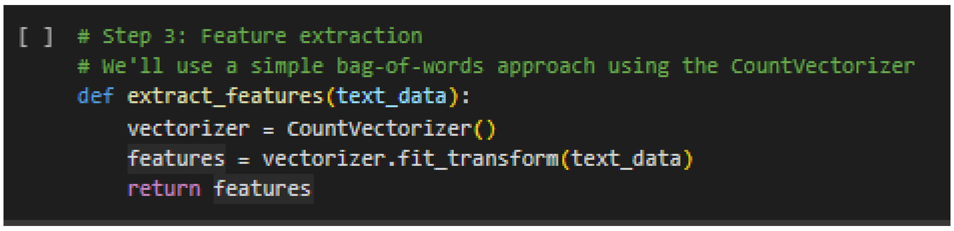
```
# Assuming the CSV file has two columns: 'url' and 'is_blog'
csv_file_path = './webpageDataset.csv'

# Step 1: Data prepare
# Create empty lists to store the data
web_pages = []
labels = []
# Read data from the CSV file
with open(csv_file_path, 'r') as file:
    reader = csv.DictReader(file)
    for row in reader:
        web_pages.append(row['url'])
        labels.append(row['is_blog'])
```

- This function uses the request library to send an HTTP GET request to the specified URL and obtain the web page content. The requests.get(url) method returns a Response object, which contains the server's response.
- The BeautifulSoup is then created, passing the response.content and specify the parser to use ('html.parser'). The BeautifulSoup library allows us to parse and extract data from HTML document.



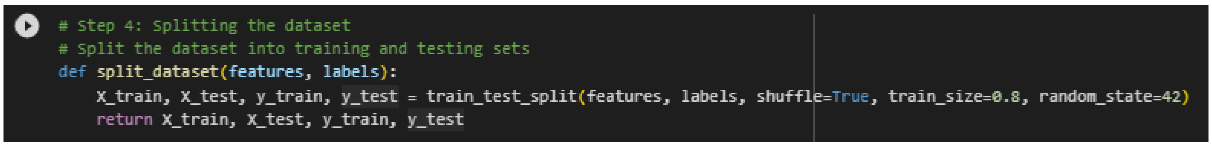
```
[ ]  # Step 2: Data preprocessing
     # Preprocess the web page data to extract the text content
     def preprocess_web_page(url):
         response = requests.get(url)
         soup = BeautifulSoup(response.content, 'html.parser')
         # Extract the text content from the web page
         text = soup.get_text()
         return text
```

- This code defines a function that uses the `CountVectorizer` class to convert text data into a numerical feature matrix. The feature matrix can then be used for further analysis or as input to machine learning algorithms.



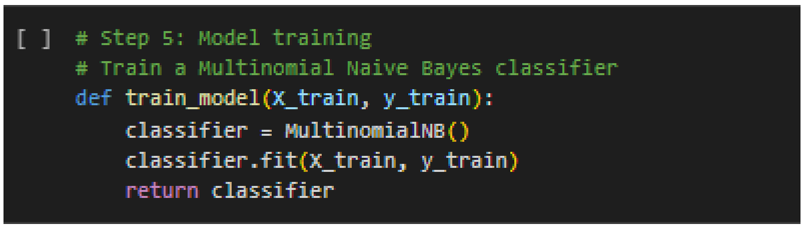
```
[ ]  # Step 3: Feature extraction
     # We'll use a simple bag-of-words approach using the CountVectorizer
     def extract_features(text_data):
         vectorizer = CountVectorizer()
         features = vectorizer.fit_transform(text_data)
         return features
```

- This function splits the dataset into training and testing sets using the specified proportions and random seed. It returns the split data, allowing further analysis or modeling on the separate sets.



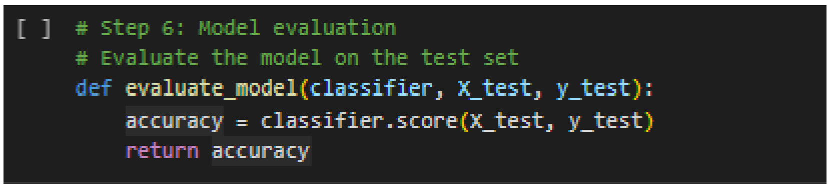
```
# Step 4: Splitting the dataset
# Split the dataset into training and testing sets
def split_dataset(features, labels):
    X_train, X_test, y_train, y_test = train_test_split(features, labels, shuffle=True, train_size=0.8, random_state=42)
    return X_train, X_test, y_train, y_test
```

- This function trains a Multinomial Naive Bayes classifier on the provided training data (features and labels) using the fit method. The trained classifier is returned for further use, such as making predictions on new data.



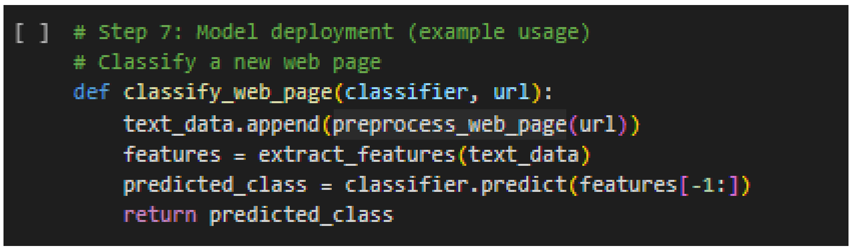
```
[ ]  # Step 5: Model training
     # Train a Multinomial Naive Bayes classifier
     def train_model(X_train, y_train):
         classifier = MultinomialNB()
         classifier.fit(X_train, y_train)
         return classifier
```

- This function evaluates the performance of a trained classifier on the provided testing data by calculating the accuracy. The accuracy value is returned as the evaluation result.



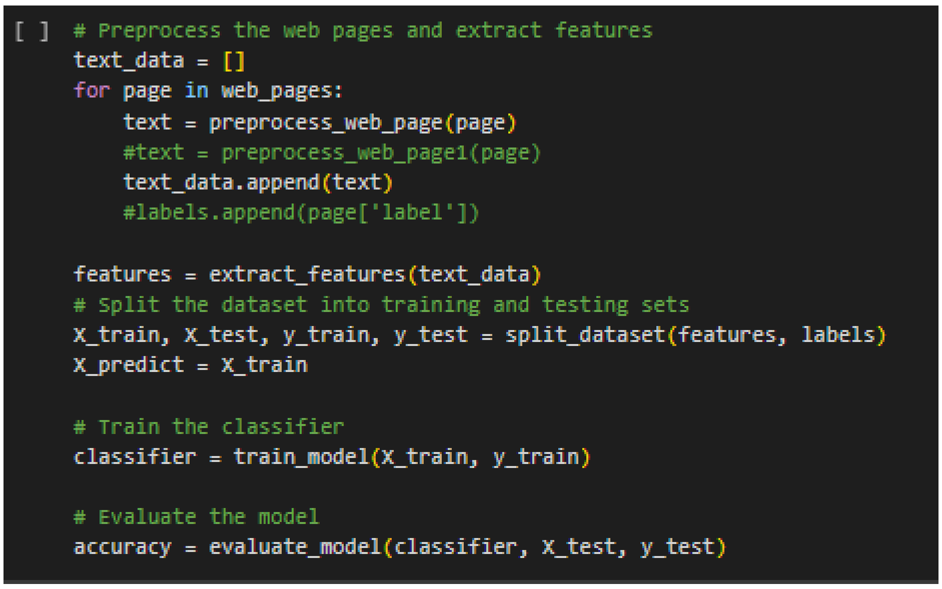
```python
# Step 6: Model evaluation
# Evaluate the model on the test set
def evaluate_model(classifier, X_test, y_test):
    accuracy = classifier.score(X_test, y_test)
    return accuracy
```

- This function classifies a given web page by preprocessing its text content, extracting numerical features, and using a trained classifier to predict its class. The predicted class is returned as the classification result.



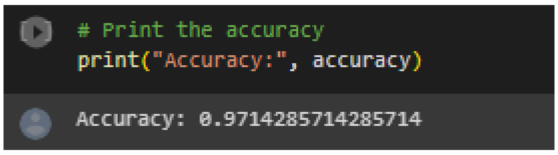
```python
# Step 7: Model deployment (example usage)
# Classify a new web page
def classify_web_page(classifier, url):
    text_data.append(preprocess_web_page(url))
    features = extract_features(text_data)
    predicted_class = classifier.predict(features[-1:])
    return predicted_class
```

- This code preprocesses the text data from the web pages, converts it into numerical features, splits the data into training and testing sets, trains a classifier, and evaluates its accuracy. The resulting accuracy is stored in the accuracy variable.



```python
# Preprocess the web pages and extract features
text_data = []
for page in web_pages:
    text = preprocess_web_page(page)
    #text = preprocess_web_page1(page)
    text_data.append(text)
    #labels.append(page['label'])

features = extract_features(text_data)
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = split_dataset(features, labels)
X_predict = X_train

# Train the classifier
classifier = train_model(X_train, y_train)

# Evaluate the model
accuracy = evaluate_model(classifier, X_test, y_test)
```

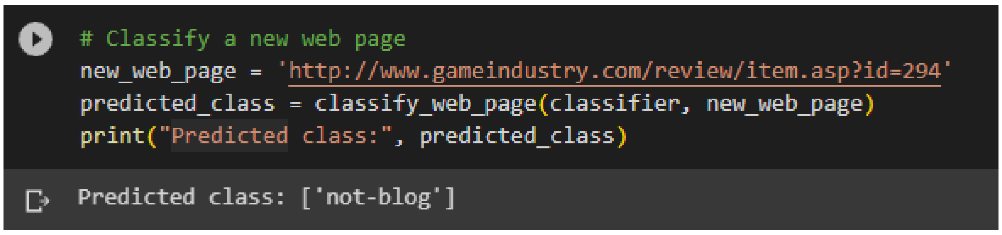
- This code printed the accuracy.



```python
# Print the accuracy
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9714285714285714
```

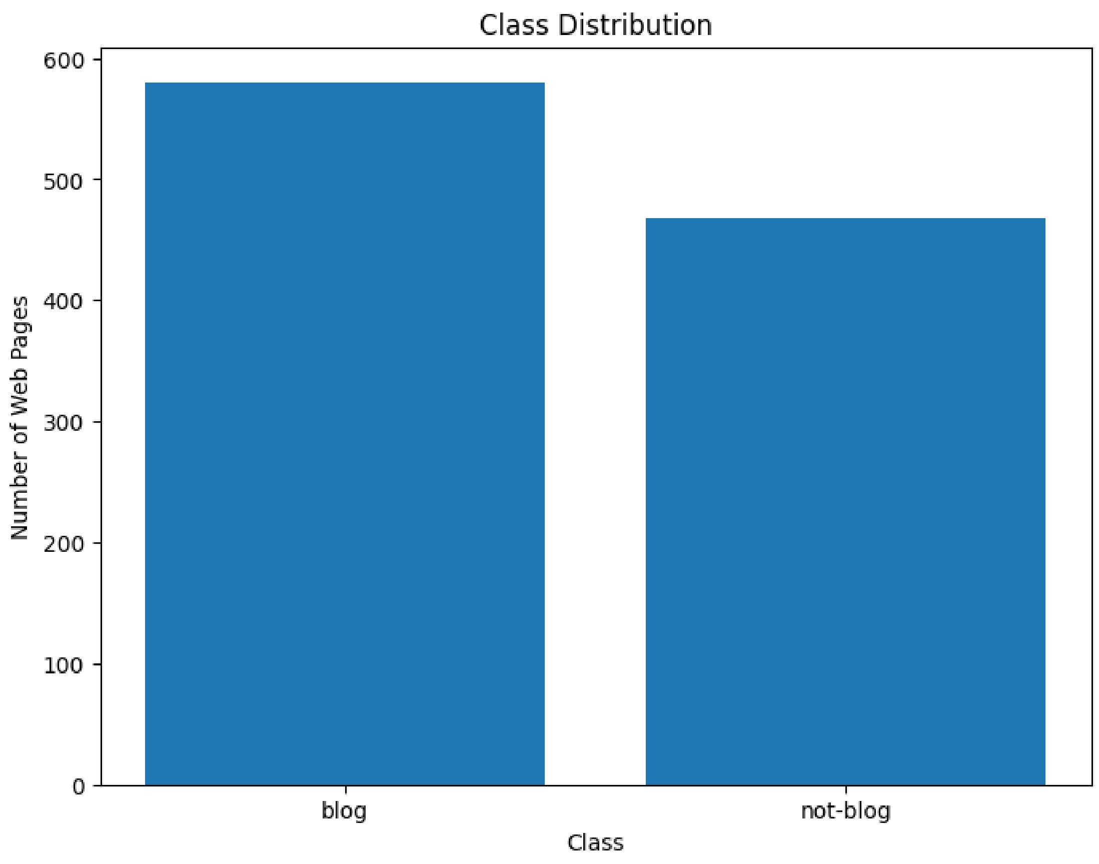
- This code snippet uses the trained classifier to predict the class of a new web page specified by the new_web_page URL and prints the predicted class.

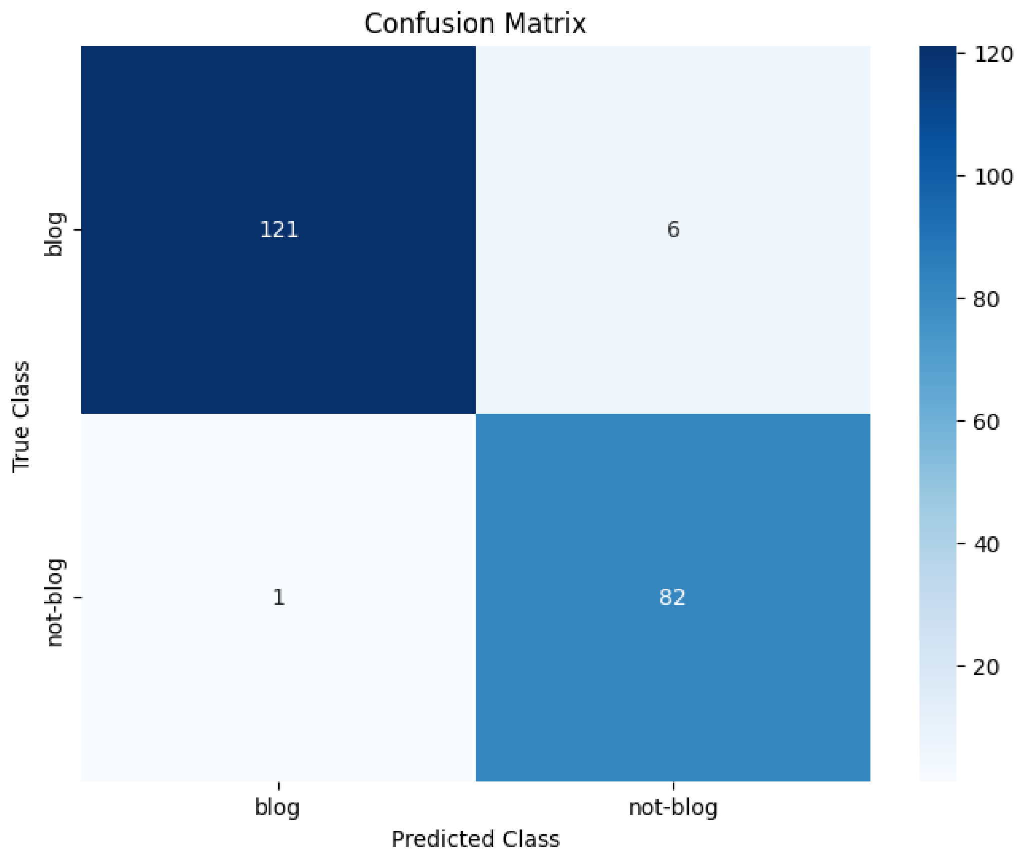


```
# Classify a new web page
new_web_page = 'http://www.gameindustry.com/review/item.asp?id=294'
predicted_class = classify_web_page(classifier, new_web_page)
print("Predicted class:", predicted_class)
```

```
Predicted class: ['not-blog']
```

Data Visualization and Outputs:




Confusion Matrix
True Class
Predicted Class
blog
not-blog
121
6
1
82
120
100
80
60
40
20