News Classification based on Their Headlines

Tan Bunchhay[#1], Tan Seyhak[#2], Bou Phally[#3], Heang seanghak[#4],
Tep Chansophanith[#5], Chanpiseth Chap[*6]

#Department of Information Technology Engineering, FE, Royal University of Phnom Penh
Phnom Penh, Cambodia

[1]2820.tan.bunchhay@rupp.edu.kh

Corresponding author: Chanpiseth Chap
[6]chap.chanpiseth@rupp.edu.kh

1 Introduction

Text mining is a very important part of natural language processing (NLP), and people have been interested in it a lot lately. The text that can be found in sources usually comes in a lot of different formats and is not organised. A lot of study papers have suggested different ways to organise this messy text into a clear, organised format that can be used for a variety of reasons, such as classifying the text. There is no doubt that using full and long texts for classification gives more accurate results than using short texts, but it takes a lot of time to do the calculations. For this project, we want to look into three ways to sort texts into groups using short texts [1], which are the text's headlines or titles. When sentences are too short, they may not give enough information, which affects the accuracy of the classifiers. However, news titles may give a lot of information about the semantic material in a short space of time.

2 Objective

The goal of this task is to use three different machine learning methods—Decision Tree, Multinomial Naive Bayes, and Artificial Neural Network—to sort news stories into groups based on their headlines. After that, the results of the three models will be closely looked at and demonstrated.

3 Scraping Data for Testing Classification

We had used Selenium for Python (https://pypi.org/project/selenium/) to scrape news articles on websites. The Python script using the Selenium library, detailed below, exemplifies this approach by scraping news headlines from a website.

3.1 Initialization of the Web Browser:

The Python script employs Selenium's webdriver to initiate a browser session. Here, the Edge browser is used. The script then accesses the target URL: https://en.freshnewsasia.com/index.php/en/.

3.2 Data Extraction Process:

Utilizing selenium.webdriver.common.by.By, the script locates web elements systematically.
browser.implicitly_wait(20) ensures the script waits up to 20 seconds for the web elements to load completely.
The key operation involves locating elements matching the CSS selector 'td.list-title a' through browser.find_elements(By.CSS_SELECTOR, 'td.list-title a'), which are likely the links to news headlines.

3.3 Data Processing and Storage:

The script extracts the textual content from each element, which are the news headlines in this context.
These headlines are stored in a CSV file named fresh-news.csv. The file creation includes writing a header, followed by appending each headline as a separate entry.
CSV writing operations are conducted using `csv.DictWriter` and `csv.writer`, ensuring proper data structuring and formatting.

3.4 Termination of the Scraping Session:

The process concludes with the closure of the CSV file using `file.close()` and termination of the browser session with `browser.quit().`

3.5 Importance in Classification Testing:

Web scraping scripts like this one are instrumental in gathering diverse and up-to-date data, which is essential for testing and training classification models in machine learning. The extracted headlines can be used as a dataset for various classification tasks, such as sentiment analysis, topic categorization, or trend identification. The ability to automate the collection of such targeted data makes web scraping an invaluable tool in the data scientist's toolkit.

4 Requirement Libraries

- pandas
- scikit-learn
- numpy
- seaborn
- jupyter
- nltk

```
#import importance libraries
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn import set_config
import nltk
import re
import string
```

5 Read Dataset to data frame

```
dataframe = pd.read_csv(src_filepath, encoding="utf8",
usecols=['TITLE', 'CATEGORY']) dataframe.column

# Display the dataset
Dataframe
```

OUTPUT:

```
TITLE CATEGORY
  …          …
```

6 Data Preprocessing

```
# Preprocessing #check for missing data
if(any(dataframe.isnull().any())):
 print('Missing Data\n')
 print(dataframe.isnull().sum())
else:
 print('NO missing data')
```

OUTPUT:

```
NO missing data
```

```
# check for duplicate
if(any(dataframe.duplicated())==True):
print('Duplicate rows found')
print('Number of duplicate rows= ',
dataframe[dataframe.duplicated()].shape[0])
    dataframe.drop_duplicates(inplace=True,keep='first')
dataframe.reset_index(inplace=True,drop=True)
    print('Dropping duplicates\n')
print(dataframe.shape) else:
print('NO duplicate data')
```

OUTPUT:

```
Duplicate rows found
Number of duplicate rows=  4866
Dropping duplicates
(132799, 2)
```

```
# download the library to for the nltk functions to use in the
cleaning process nltk.download('stopwords')
nltk.download('punkt') nltk.download('wordnet')

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
set_config(transform_output="pandas")

wnl = WordNetLemmatizer()


# Function for cleaning and tokenize the headline def tokenize(doc):
document = doc.lower()
    document = re.sub(r'\d+', '',document)
    document = document.translate(str.maketrans('', '', string.punctuation))
    document = document.strip()
    return [wnl.lemmatize(token) for token in word_tokenize(document)
    if token not in stopwords.words('english')]

# The preprocess pipeline
preprocessor = Pipeline([
    ('vect', CountVectorizer(tokenizer=tokenize)),
    ('tfidf', TfidfTransformer()),
])

tfidf_dataset = preprocessor.fit_transform(dataframe["TITLE"].values)
```

7 Model Implementation and Evaluation

7.1 Label Encoder

```
le = LabelEncoder()
class_label = le.fit_transform(dataframe["CATEGORY"])
# list(le.classes_)
class_label
array([0, 0, 0, ..., 1, 1, 1])
X_train, X_test, y_train, y_test = train_test_split(
tfidf_dataset.toarray(),     class_label,
test_size = 0.3,     random_state=42
)
```

7.2 Decision Tree Classifier

```
dt_classifier = DecisionTreeClassifier(criterion="gini",
splitter="best")
dt_classifier.fit(X_train, y_train) dt_predictions =
dt_classifier.predict(X_test)

# Evaluation
print("accuracy score of Decision Tree:")
print(accuracy_score(y_test, dt_predictions))
```

OUTPUT:

```
accuracy score of Decision Tree: 0.9151606425702812
```

7.3 Multinomial Naive Bayes Classifier

```
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)
nb_predictions = nb_classifier.predict(X_test)

# Evaluation
print("accuracy score of Multinomial Naive Bayes:")
print(accuracy_score(y_test, nb_predictions))
```

OUTPUT:

```
accuracy score of Multinomial Naive Bayes:
0.9354166666666667
```

7.4 Artificial Neural Network

```
nn_classifier = MLPClassifier()
nn_classifier.fit(X_train, y_train)
nn_predictions = nn_classifier.predict(X_test)

# Evaluation
print("accuracy score of Artificial Neural Network:")
print(accuracy_score(y_test, nn_predictions))
```

OUTPUT:

```
accuracy score of Artificial Neural Network:
0.9492720883534137
```

7. Conclusion

In this project, we explored three machine learning models - Decision Tree, Multinomial Naive Bayes, and Artificial Neural Network - for news headline classification. Each model brought unique strengths to the task:

- The Decision Tree likely offered a clear, interpretable model but might have faced limitations in handling complex text data.
- The Multinomial Naive Bayes, often effective for text classification, probably performed well due to its simplicity and efficiency.
- The Artificial Neural Network, with its advanced pattern recognition capabilities, was expected to provide the highest accuracy.

While each model has its advantages, the choice often depends on the specific requirements and nature of the dataset. Future work could focus on further tuning these models, exploring advanced techniques like deep learning, and experimenting with different feature extraction methods for improved accuracy.