import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import nltk

from nltk.corpus import stopwords

nltk.download('stopwords')

from nltk.stem import PorterStemmer

from nltk.corpus import stopwords

import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)

Importing necessary libraries: pandas, numpy, matplotlib.pyplot, and additional libraries related to text processing such as nltk (Natural Language Toolkit) for text preprocessing and warnings to suppress future warnings.

python

Copy code

import pandas as pd

import numpy as np

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn import preprocessing

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import GridSearchCV

Importing libraries for machine learning tasks: CountVectorizer and TfidfVectorizer for text feature extraction, LogisticRegression for logistic regression classification, train\_test\_split for splitting data into training and testing sets, accuracy\_score for evaluating classification accuracy, preprocessing for label encoding, RandomForestClassifier for random forest classification, and GridSearchCV for hyperparameter tuning.

python

Copy code

data = pd.read\_csv("bbc-news-data.csv", sep = "\t")

df = data.drop("filename" , axis=1)

Reading the data from a CSV file named "bbc-news-data.csv" and storing it in a DataFrame called data. Then, dropping the "filename" column from the DataFrame and assigning the result to a new DataFrame df.

python

Copy code

df['title'] = df['title'].str.lower()

df['content'] = df['content'].str.lower()

df['category'] = df['category'].str.lower()

Converting the text in the "title", "content", and "category" columns to lowercase using the str.lower() method.

python

Copy code

print(data['category'].unique())

Printing the unique categories present in the "category" column of the original data DataFrame.

python

Copy code

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

df['category'] = le.fit\_transform(df['category'])

Importing LabelEncoder from sklearn.preprocessing to encode the categories in the "category" column of df DataFrame into numerical labels. Fit-transforming the categories using fit\_transform() and replacing the "category" column with the encoded labels.

python

Copy code

import re

df['content'] = df['content'].apply(lambda x: re.sub(r'[^a-zA-Z0-9\s]', '', x))

Removing special characters and symbols from the text in the "content" column using regular expression (re.sub()).

python

Copy code

df['content'] = df['content'].apply(lambda x: ' '.join(word for word in x.split() if len(word) > 1))

df

Removing single-character words from the text in the "content" column using a lambda function and list comprehension.

python

Copy code

stop\_words = set(stopwords.words('english'))

df['title'] = df['title'].apply(lambda x: ' '.join([word for word in x.lower().split() if word not in stop\_words]))

print(df.head())

Importing the English stop words from the NLTK library and storing them in a set called stop\_words. Removing the stop words from the text in the "title" column using a lambda function and list comprehension. Printing the first few rows of the modified DataFrame.

python

Copy code

ps = PorterStemmer()

df['content'] = df['content'].apply(lambda x: ' '.join([ps.stem(word) for word in x.lower().split()]))

print(df.head())

Initializing a PorterStemmer object for word stemming. Applying stemming to the words in the "content" column using a lambda function and list comprehension.

python

Copy code

stop\_words = (stopwords.words('english')) # список стоп-слов

vectorizer = TfidfVectorizer(stop\_words=stop\_words)

X = vectorizer.fit\_transform(df.content.values.astype('U'))

y = df['category'].values

Reimporting the English stop words from the NLTK library and storing them in a list called stop\_words. Creating a TfidfVectorizer object and passing the stop words as an argument. Vectorizing the text in the "content" column using fit\_transform() and assigning the result to X. Assigning the "category" column as the target variable y.

python

Copy code

le = preprocessing.LabelEncoder()

le.fit(df.category)

df.category = le.transform(df.category)

y = df.category

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=40)

Reimporting the LabelEncoder from sklearn.preprocessing and encoding the "category" column of the DataFrame df using fit() and transform(). Assigning the encoded labels back to the "category" column. Splitting the data into training and testing sets using train\_test\_split() with 33% of the data allocated for testing and a random state of 40. Assigning the resulting training and testing sets to X\_train, X\_test, y\_train, and y\_test.

python

Copy code

logisticRegr = LogisticRegression()

logisticRegr.fit(X\_train, y\_train)

y\_pred = logisticRegr.predict(X\_test)

accuracy\_score(y\_pred, y\_test)

logisticRegr.score(X\_test, y\_test)

Creating a logistic regression classifier object called logisticRegr and fitting it to the training data using fit(). Predicting the categories for the testing data using predict(). Evaluating the accuracy of the predictions using accuracy\_score(). Calculating the accuracy score directly using the score() method.

python

Copy code

rf = RandomForestClassifier(n\_estimators=100, max\_features=5,

max\_depth=None, min\_samples\_leaf=1, random\_state=40)

Creating a random forest classifier object called rf with hyperparameters: 100 estimators, 5 maximum features, no maximum depth limit, minimum samples per leaf as 1, and a random state of 40.

python

Copy code

print(validation\_curve(rf, X\_train, y\_train, param\_name="max\_depth",

param\_range=np.arange(1, 11), cv=10, scoring="accuracy"))

Using the validation\_curve() function from yellowbrick.model\_selection to perform a validation curve analysis on the random forest classifier with varying maximum depths. The range of maximum depths is specified as 1 to 10, with cross-validation of 10 folds, and accuracy as the scoring metric.

python

Copy code

print(validation\_curve(rf, X\_train, y\_train, param\_name="n\_estimators",

param\_range=np.arange(1, 11), cv=10, scoring="accuracy"))

Using the validation\_curve() function again, this time to perform a validation curve analysis on the random forest classifier with varying numbers of estimators. The range of estimators is specified as 1 to 10, with cross-validation of 10 folds, and accuracy as the scoring metric.