MACHINE LEARNING PROJECT EXAMS

Name : Amanda Ofori

Course: BSc. Computer Science

Roll Number: 10201100146

Question 2: **News Categorizing System - The Documentation**

Link to Git Repo: https://github.com/Amanda-Ofori/Better-News-Article-Classifier.git

Backround of News Classifiction System

The swift expansion of digital media has resulted in an immense flood of news pieces across multiple platforms. Because there is so much data available, news items needs to be accurately and efficiently categorized. In order to meet this need, news classification algorithms are essential since they automatically group items into pre-established categories like business, politics, sports, and entertainment. These approaches facilitate simpler access to pertinent information and offer personalized news feeds, which improve user experience in addition to helping with content management.

The various ways that news classification models are used in the media and information sector highlight how important they are. These approaches provide the automatic tagging and archiving of articles, hence simplifying content management procedures for publishers and news aggregators. In order to meet this need, news classification algorithms are essential since they automatically group items into pre-established categories like business, politics, sports, and entertainment. These replicas don't Readers can locate interesting articles more quickly thanks to classified news articles' more focused and organized reading experience. Furthermore, news classification models play a key role in personalized news recommendation systems, which employ the categorized content to provide consumers with individualized news feeds according to their reading history and interests.

Introduction

This report's main goal is to provide documentation of the techniques used to create a highly efficient system for classifying news articles that makes use of the information found in published articles to create an improved model. The document functions as a thorough road map, detailing the methods and procedures used to arrive at a 99% accurate news classification model.

Text classification is an important task in natural language processing (NLP) and machine learning, particularly when it comes to news article classification into different categories. This project is important because it can automatically organize large volumes of textual data, which will improve content analysis, information retrieval, and news distribution efficiency.

This paper, which draws inspiration from a plethora of research and insights from the literature already in existence, synthesizes the best practices and creative techniques to handle the difficulties associated with classifying news articles. The objective is to offer a comprehensive manual that covers feature engineering, data preprocessing, model selection, training, evaluation, and deployment in order to produce a model that is not only highly accurate but also resilient and flexible enough to work with a variety of news datasets.

This report intends to be a useful tool for researchers, data scientists, and practitioners in the field of natural language processing and machine learning by carefully outlining the techniques and insights obtained from published articles. It will assist them in their endeavor to develop a news classification model that is 99% accurate.

Literature Review

Article 1 Title: A News Story Categorization System

Authors: Philip J. Hayes, Laura E. Knecht, Monica J. Cellio

Institution: Carnegie Group Inc, Pittsburgh, PA

In order to classify news stories into broad topic categories, a commercial application that uses natural language processing algorithms is presented in this study in prototype form. To detect pieces indicative of particular categories, the system uses knowledge-based rules and pattern-matching approaches. Without carrying out extensive syntactic or semantic analyses, it attains an accuracy that is marginally lower than that of human categorization

A system for grouping news articles into broad topic groups has been developed as a result of the growing interest in the commercial uses of computerized text processing. Human labor has historically been used for this, which is costly, sluggish, and inconsistent. The speed, cost, and consistency of text processing can all be improved with automatic systems.

The objective was to show that computer-based classification of news articles utilizing natural language processing methods is feasible. In order to simulate human categorization, the machine had to group stories into six of the 72 available categories. The ambiguity and subjectivity of some categories as well as the inconsistent nature of human classifications presented difficulties.

There are two stages to the system's operation: hypothesis and confirmation. Identifying possible categories based on the existence of specific words and phrases is known as hypothesising. Verifying these categories using more data and rules akin to those of an expert system is the process of confirmation. The approach makes use of patternsets, which are groups of words and phrases connected to particular ideas.

After accounting for human categorization discrepancies, the system tested on 500 randomly selected stories and averaged 93% accuracy for both recall and precision. On a Symbolics 3640 system, this was accomplished with an average processing time of 15 seconds per narrative.

Without requiring in-depth linguistic investigations, pattern-matching algorithms and knowledge-based rules can be used to achieve high accuracy in automatic text categorization. There is a lot of business possibilities for text archiving and routing with this.

Article 2 Title: Building a Text Classification System for News Articles: A Comprehensive Guide

Author: Jyotsna Pyarasani

Date: January 12, 2024

Text classification is the process of classifying textual input into predetermined categories using natural language processing (NLP). This tutorial covers the steps involved in developing a text categorization system for news items, including importing data, preprocessing, training models, and deploying the system.

Step 1: Loading Data

To load news articles into a Pandas DataFrame from various categories (business, entertainment, politics, sport, and tech), use a custom function called load_data.

Step 2: Preprocessing and Data Cleaning

To prepare text data for machine learning algorithms, convert text to lowercase, tokenize into words, remove punctuation and stopwords, and apply stemming.

Step Three: Engineering Features

Utilize methods such as Term Frequency-Inverse Document Frequency (TF-IDF) and Bag-of-Words (BoW) to transform textual data into a format that is appropriate for machine learning models.

Step Four: Instruction of Models

Examine and train a variety of machine learning models, such as Random Forest and Naive Bayes, for text classification using the feature-engineered data.

Step 5: Assessment of the Model

To assess the trained models' performance on the test set, use measures like confusion matrix, accuracy, and precision.

Creating an Ensemble Model in Step Six

Step Four: Instruction of Models

Examine and train a variety of machine learning models, such as Random Forest and Naive Bayes, for text classification using the feature-engineered data.

Step 5: Assessment of the Model

To assess the trained models' performance on the test set, use measures like confusion matrix, accuracy, and precision.

Creating an Ensemble Model in Step Six

To assess the performance of the ensemble model and enhance overall performance, combine many models using a Voting Classifier.

Step 7: Utilizing the Model

Using the pickle library, store the model that performs well for use in practical situations.

Step 8: Developing a Clear App

To communicate with the trained model, develop a Streamlit web application that lets users enter a text message and guess the news article's category.

In summary, this book shows how machine learning may be used to process and comprehend textual data by offering a framework for creating a text classification system for news articles. The sample can be developed upon for certain use cases and serves as a foundation for more intricate NLP projects.

THE CODE DOCUMENTATION OF BETTER-NEWS ARTICLE CLASSIFIER APP

Importing Packages

Importing of the neccessary libraries for the news categorizing app highly required. Here is a little information about the libraries imported;

1. `pandas` : This is a data analysis and manipulation library. It provides procedures and data structures for working with time series and numerical tables.
2. **{nltk}**: The Natural Language Toolkit is a library for manipulating text containing human language. It offers a set of text processing packages for categorization, tokenization, stemming, tagging, parsing, and semantic reasoning in addition to user-friendly interfaces for more than 50 corpora and lexical resources.
3. `stopwords` : A module from the NLTK library that eliminates words that don't add much to the meaning of a document, such as "and," "the," etc.
4. `WordNetLemmatizer` : an NLTK lemmatizer that searches the WordNet Database for word lemmas. Lemmatization is the process of organizing a word's various inflected forms into a single group so that they can be examined as a unit.

5. `word_tokenize` : an NLTK function that divides text into tokens, which are typically words or phrases.

1. `TfidfVectorizer` : From scikit-learn's `feature_extraction.text` , it transforms text to feature vectors that can be used as input to estimator. TF-IDF stands for term-frequency times inverse document-frequency, which is a way to normalize term counts by considering the number of documents that include the word.

2. `GridSearchCV` : a scikit-learn module for adjusting a model's hyperparameters. It looks for an estimator by thoroughly searching over a range of parameter values.

3. `train_test_split` : A scikit-learn function that divides matrices or arrays into random train and test subsets.

4. `VotingClassifier` : a classifier from scikit-learn that predicts class labels by averaging predicted probabilities or a majority vote between conceptually distinct machine learning classifiers.

5. `cross_val_score` : A scikit-learn function that uses cross-validation to assess a score.

6. `MultinomialNB` : A scikit-learn Naive Bayes classifier appropriate for discrete feature classification (word counts for text classification, for example).

7. `RandomForestClassifier` : a kind of ensemble machine learning classifier that employs averaging to increase prediction accuracy and manage over-fitting while using multiple decision tree classifiers on different dataset subsamples.

8. `XGBClassifier` :A gradient boosted decision tree implementation from the XGBoost package that is optimized for speed and performance.

9. `accuracy_score` , `confusion_matrix` , `classification_report` : Scikit-learn functions that assess how well a classification algorithm predicts the future. The collection of labels predicted for a sample that must precisely match the corresponding set of labels in {y_true} is determined by the accuracy score.

10. `requests` : A library for submitting many types of HTTP requests. It is employed to send HTTP requests to a web server.

11. `re` : Regular expression matching operations akin to Perl's are provided by this Python package.

12. `os` : a module that offers a portable method to use functions like reading and writing to files that are dependent on the operating system.

13. `pickle` : An object in Python can be converted into a byte stream and then back again with the help of this module, which can serialize and deserialize object structures.

14. `Counter` : A container that tracks the number of times equivalent values are inserted, from the {collections` module. It can be applied to swiftly and effectively develop counting algorithms.

This information is coined from credible resources;

- Scikit-learn. (2019). scikit-learn: machine learning in Python — scikit-learn 0.20.3 documentation. Scikit-Learn.org.
- Payne, R. (2023, September 22). 7 Best Python Libraries for Machine Learning and AI. Developer.com.

In [ ]:
```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
import requests
import re
import os
import pickle
from collections import Counter
```

Setting up Directory to store models

In this section of the code, it is meant to control the process of creating a directory on a filesystem where models are kept. Here are the steps that is shown in the codes;

- Choose the Directory Name: The directory name where models should be kept is indicated by the variable models_dir, which is set to the string "models".
- Verify Directory Existence: The if not os.path.exists(models_dir) line verifies whether the filesystem contains a directory with the name given in models_dir.
- Establish Directory: The following code runs inside the if block if the directory is not there (that is, if os.path.exists(models_dir) evaluates to True):
  - Models_dir = os.makedirs(): The directory named models is created by this function.
  - output(f"Created directory: {models_dir}"): This creates the directory and prints a message indicating that.
- Directory Exists: The function inside the else block runs if the directory already exists (or if os.path.exists(models_dir) evaluates to False).
  - print(f"Directory already exists: {models_dir}"): This outputs a notification noting the existence of the directory.

In [ ]:
```
# Specify the directory name where you want to store the models
models_dir = "models"

# Create the directory if it does not exist
if not os.path.exists(models_dir):
    os.makedirs(models_dir)
```

```
    print(f"Created directory: {models_dir}")
else:
    print(f"Directory already exists: {models_dir}")
```

```
Created directory: models
```

Loading the Dataset and Defining EDA and Preprocessing functions

This snippet of code performs a number of preparation, data loading, and exploratory actions on a dataset of news article categories. Here is some further insights into the code snippet;

The dataset should be loaded as follows: data = pd.read_csv('news-article-categories.csv', encoding='latin-1'): This line loads the 'news-article-categories.csv' CSV file into the data pandas DataFrame. The character encoding used to read the file is specified by the encoding='latin-1' option, which is for handling special characters in the dataset.

Print the Exploration Notice and Initial Data: print("EDA on News Category dataset"): Outputs a notice that the news category dataset will soon be the subject of an Exploratory Data Analysis (EDA). print(data.head()): Provides a brief overview of the structure and contents of the dataset by displaying the first five rows of the DataFrame. Prints with print("*"15):

Information Summary for Data:

print(data.info()): This function provides a brief summary of the DataFrame's attributes, such as the total number of columns, the number of entries, the data type of each column, and the number of non-null values in each column.

A Synopsis of Numerical Column Statistics:

print(data.describe()): Provides a summary of the data, including the mean, standard deviation, etc

Missing Value Count:

data = missing_values.not null().sum(): Determines how many missing (NaN) values there are in each DataFrame column, then counts them and stores the result in the missing values variable.

Managing Missing Data

body[data] = body[data].fillna(''): Inserts empty strings ('') in place of missing values (NaN) in the DataFrame's 'body' column. print(data.columns): Prints the names of the DataFrame's columns for validating

Next is the preprocessing

In order to prepare article text for machine learning tasks, this code snippet shows a setup for preprocessing text data within the dataset:

- nltk.download('punkt'), nltk.download('stopwords'), nltk.download('wordnet'): downloading NLTK resources These lines retrieve necessary resources for text tokenization, stopword removal, and word lemmatization from the Natural Language Toolkit (NLTK).

- Set lemmatizer = WordNetLemmatizer(): This function generates a WordNetLemmatizer instance for the purpose of lemmatizing words into their root or base form.

- Every defined function carries out particular text manipulations:

  - lemmatize_words(text): Lemmatizes all words in the text to their most basic forms.
  - remove_urls(text): Extracts the text's URLs.remove_single_characters_and_excessive_spaces(text): This function cleans up the text by getting rid of isolated single characters and additional spaces.
  - convert_lower(text): Ensures consistency and avoids case-sensitive processing problems by converting all of the text's characters to lowercase.
  - remove_special_chars(text): Removes special characters and only keeps alphanumeric characters in the text.
  - remove_tags(text): Removes all potential HTML or XML tags from the content.
  - remove_stopwords(text): Eliminates frequently used terms (such as "and," "the," etc.) that don't add anything to the text's meaning.
  - Mode(results): This function for summarizing or aggregating text processing results since it finds the category or result that occurs the most frequently in a collection.

  Function of Composite Preprocessing:

  preprocess_article_text(article_text): This function carries out the cleaning steps one after the other, starting with deleting HTML tags and ending with lemmatizing the words, in order to fully process the article text.

Apply Dataset Preprocessing:

body[data] = body[data].apply the preprocessed article text. applies the preprocess_article_text function to every element in the data DataFrame's 'body' column, thereby standardizing and thoroughly cleaning the textual information for every article.

Names of Print Columns:

print(data.columns): This function shows the DataFrame's column names, to confirm the dataset's structure followed the preprocessing.

In [ ]:
```
# Load the data set
data = pd.read_csv('news-article-categories.csv', encoding='latin-1')

print("EDA on News Category dataset")
print(data.head())
print("**"*15)
# Get a concise summary of the dataframe
print(data.info())
print("**"*15)
# Summary statistics for numeric columns
print(data.describe())
print("**"*15)
# Count the number of missing values in each column
```

```
missing_values = data.isnull().sum()
print(missing_values)


# Fill NaN values in the 'body' column with empty strings
data['body'] = data['body'].fillna('')
print(data.columns)

# Preprocessing of the dataset

 # Ensure you've downloaded necessary NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet') # For lemmatization

# Initialize the WordNet Lemmatizer
lemmatizer = WordNetLemmatizer()

# Define preprocessing functions
def lemmatize_words(text):
    wordnet = WordNetLemmatizer()
    words = word_tokenize(text)
    return " ".join([wordnet.lemmatize(word) for word in words])

def remove_urls(text):
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    return text

def remove_single_characters_and_excessive_spaces(text):
    # Remove single characters and excessive spaces
    text = re.sub(r'\s+[a-zA-Z]\s+', ' ', text)  # Remove single characters surrour
    text = re.sub(r'^[a-zA-Z]\s+', ' ', text)    # Remove single character at the s
    text = re.sub(r'\s+', ' ', text, flags=re.I) # Remove excessive spaces
    return text.strip()  # Remove leading and trailing spaces

def convert_lower(text):
    return text.lower()

def remove_special_chars(text):
    text_without_special_chars = []
    for char in text:
        if char.isalnum():
            text_without_special_chars.append(char)
        else:
            text_without_special_chars.append(' ')
    return "".join(text_without_special_chars)

def remove_tags(text):
    tag_regex = re.compile(r'<[^>]+>')
    return re.sub(tag_regex, '', text)

def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(text)
    return " ".join([word for word in words if word not in stop_words])

def mode(results):
    freqs = Counter(results)
    ranking = sorted(freqs.items(), key=lambda pair: pair[1], reverse=True)
    return ranking[0][0]  # This returns the most frequent category name directly


# Define preprocessing functions
```

```
def preprocess_article_text(article_text):
    article_text = remove_tags(article_text)
    article_text = remove_special_chars(article_text)
    article_text = convert_lower(article_text)
    article_text = remove_stopwords(article_text)
    article_text = lemmatize_words(article_text)
    return article_text


# pre-process the text from the dataset
data['body'] = data['body'].apply(preprocess_article_text)
print(data.columns)
```

```
EDA on News Category dataset
         category                                                 title  \
0  ARTS & CULTURE  Modeling Agencies Enabled Sexual Predators For...
1  ARTS & CULTURE  Actor Jeff Hiller Talks â□□Bright Colors And B...
2  ARTS & CULTURE  New Yorker Cover Puts Trump 'In The Hole' Afte...
3  ARTS & CULTURE  Man Surprises Girlfriend By Drawing Them In Di...
4  ARTS & CULTURE  This Artist Gives Renaissance-Style Sculptures...

                                                body
0  In October 2017, Carolyn Kramer received a dis...
1  This week I talked with actor Jeff Hiller abou...
2  The New Yorker is taking on President Donald T...
3  Kellen Hickey, a 26-year-old who lives in Huds...
4  Thereâ□□s something about combining the tradit...
******************************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6877 entries, 0 to 6876
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   category  6877 non-null   object
 1   title     6877 non-null   object
 2   body      6872 non-null   object
dtypes: object(3)
memory usage: 161.3+ KB
None
******************************
             category                                title  \
count            6877                                 6877
unique             14                                 6836
top     ARTS & CULTURE  Extreme Weather Photos Of The Week
freq             1002                                   24

                                                    body
count                                               6872
unique                                              6815
top     This week brought several big headlines in ext...
freq                                                  21
******************************
category    0
title       0
body        5
dtype: int64
Index(['category', 'title', 'body'], dtype='object')
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
Index(['category', 'title', 'body'], dtype='object')
```

Defining the train models function

This function encapsulates the process of setting up, training, tuning, and evaluating a text classification model. Here is the explanaition;

A machine learning model pipeline for text categorization is built up and trained using the function train_models() defined in this code. This is a condensed explanation:

Data Splitting: train test split is used to separate the dataset into training and testing sets. Thirty percent of the data must be tested.

Text data can be transformed into a matrix of TF-IDF features by initializing a TfidfVectorizer. changes both training and test data by fitting this vectorizer to the training set.

Save Vectorizer: if no file is already there, the vectorizer is saved to one. Otherwise, it checks to see if a file exists.

Initialization of the Model:

XGBoost, Random Forest, and Multinomial Naive Bayes classifiers are initialized. combines these using a soft voting method to create a Voting Classifier for ensemble learning.

Training Models:

uses the vectorized training data to train the Voting Classifier.

Adjusting Hyperparameters:

utilizes GridSearchCV to adjust the Random Forest model's parameters, such as the minimum samples split, maximum depth, and number of estimators.

Retrain and Update the Voting Classifier:

replaces the current Voting Classifier with the grid search's top-performing Random Forest model. The revised Voting Classifier is retrained.

Preserve the trained model:

saves the Voting Classifier that has been taught to a file, making sure the file doesn't already exist to prevent overwriting.

Assessment of the Model:

predicts the labels for the test data and determines the accuracy. prints the precision along with a thorough classification report that includes metrics like precision, recall, and F1-score.

Return Quantities:

provides the accuracy score and the trained Voting Classifier.

In [ ]:
```
# Define the model
def train_models():
    # Split the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(data['body'], data['categor

# Initialize TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer()

# Fit the vectorizer on the training data and transform it
X_train_vectorized = tfidf_vectorizer.fit_transform(X_train)

# Transform the test data using the same vectorizer
X_test_vectorized = tfidf_vectorizer.transform(X_test)

# Save model
model_filename = os.path.join(models_dir, "vectoriser.pkl")
if not os.path.exists(model_filename):
    with open(model_filename, 'wb') as f:
        pickle.dump(tfidf_vectorizer, f)
    print(f"Vectorizer saved to: {model_filename}")
else:
    print(f"Vectorizer file already exists: {model_filename}")


# Define individual models
mnb = MultinomialNB()
rfc = RandomForestClassifier(n_estimators=50, random_state=2)
xgb = XGBClassifier(n_estimators=50, random_state=2)

# Create a Voting Classifier
voting_clf = VotingClassifier(
    estimators=[('mnb', mnb), ('rf', rfc), ('xgb', xgb)],
    voting='soft'
)

# Training the Ensemble Model
voting_clf.fit(X_train_vectorized, y_train)

# Hyperparameter Tuning for Random Forest
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=3, n_jobs=-
grid_search.fit(X_train_vectorized, y_train)
rfc_best = grid_search.best_estimator_

# Update the Voting Classifier with the best Random Forest model
voting_clf.set_params(rf=rfc_best)

# Re-train the updated Ensemble Model
voting_clf.fit(X_train_vectorized, y_train)

# Save the trained voting classifier to a file named 'voting_classifier.pkl'
model_filename = os.path.join(models_dir, "voting_classifier.pkl")
if not os.path.exists(model_filename):
    with open(model_filename, 'wb') as f:
        pickle.dump(voting_clf, f)
        print(f"Saved trained voting classifier to: {model_filename}")
else:
    print(f"File already exists: {model_filename}. Skipping save operation.")

# Model evaluation
y_pred = voting_clf.predict(X_test_vectorized)
accuracy = accuracy_score(y_test, y_pred)
```

```
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))

    return voting_clf, accuracy
```

Loading the Models for Prediction

A text vectorizer and many machine learning models are loaded from files kept in a designated directory by the function load_models_and_vectorizer, which is defined in this code. This is a brief summary of the function's capabilities:

Set up the variables: vectorizer: It will save the loaded TF-IDF vectorizer when it is initially set to None. models: Using their names as keys, the loaded models are stored in an initialized dictionary.

Directory Traversal: os.walk(models_path): Gos through the directory that models_path points to. Every file in the directory and all of its subdirectories is iterated over.

Processing of Files:

Every file discovered throughout the directory walk: os.path.join(root, file) model_path: creates the file's whole path. using f = open(model_path, 'rb'): allows for binary read mode file opening.

Vectorizer Loading and Models:

Verifies whether the file name is "vectoriser.pkl"; if so, pickle is used to load the vectorizer.sends it to the vectorizer variable using load(f). It presumes the file is a model if it is not the vectorizer: model_name is a file.split('.')[0]: Removes the file extension and extracts the model name from the file name. pickle is models[model_name].load(f): Using pickle, loads the model and stores it in the models dictionary, where the key is model_name.

Return Quantities:

The vectorizer and the models dictionary, which contains all of the loaded models, are returned by the function.

In [ ]:
```
# Load models and vectorizer for prediction
def load_models_and_vectorizer(models_path):
    vectorizer = None
    models = {}
    for root, _, files in os.walk(models_path):
        for file in files:
            model_path = os.path.join(root, file)
            with open(model_path, 'rb') as f:
                if file == 'vectoriser.pkl':
                    vectorizer = pickle.load(f)
                else:
                    model_name = file.split('.')[0]  # Extract model name from file
                    models[model_name] = pickle.load(f)
    return vectorizer, models
```

Defining the Make_Prediction function

This code defines the make_prediction function, which uses several machine learning models to categorize an article content:

Define Categories: categories: A list of possible categories that the article could fall into. Predefined labels like "ARTS & CULTURE" and "BUSINESS" are examples of this.

Set up the variables: outcomes: An empty list for each model's predictions. absolute_path: Returns the path of the directory containing the script file (**file**). models_path: To specify the path where model files are kept, this function combines absolute_path with the'models' folder.

Load the vectorizer and models:

load_models_and_vectorizer(models_path): This function loads a dictionary of trained models and the TF-IDF vectorizer from the given models_path by making a call to another function.

Vectorization and Preprocessing of Text:

preprocess_article_text(article_text): Cleans and normalizes the provided article text, among other things. text_vectorized: This function converts the preprocessed text into a format appropriate for model input (vectorized text) by using the loaded vectorizer.

Forecasts from Every Model:

Cycles through every model in trained_models: For every vectorized text, a prediction is made by each model. The results list has the first forecast result [0] appended to it.

Find the Most Frequently Assumed Prediction:

mode(results): Determines the consensus category by computing the prediction that appears the most frequently across all model outcomes.

Give back the anticipated category:

The most frequently predicted category by the ensemble of models is returned by the function, predicted_category.

In [ ]:
```
# Make prediction
def make_prediction(article_text):
    categories = [
        "ARTS & CULTURE",
        "BUSINESS",
        "COMEDY",
        "CRIME",
        "EDUCATION",
        "ENTERTAINMENT",
        "ENVIRONMENT",
    ]
    results = []
    absolute_path = os.path.dirname(__file__)
    models_path = os.path.join(absolute_path, 'models')
    vectorizer, trained_models = load_models_and_vectorizer(models_path)

    # Preprocess and vectorize the article text
    processed_article_text = preprocess_article_text(article_text)
```

```
    text_vectorized = vectorizer.transform([processed_article_text])

    # Make predictions using the loaded models
    for model_name, model in trained_models.items():
        prediction = model.predict(text_vectorized)[0]
        results.append(prediction)

    # Determine the most common prediction
    predicted_category = mode(results)
    return predicted_category  # Return the predicted category directly
```

Tracking Model Experiment

In order to enable experiment tracking MLflow, a platform for overseeing the machine learning lifecyclewas used and here is a quick summary of how it was used :

Import Statements: imports log_metric and log_param, as well as mlflow, for the purpose of tracking experiment details.

imports train_models from betternews_testmodel, presumably a model-training script. Make sure the module where train_models is defined is the one this import is correctly pointing to.

mlflow.set_experiment("Better-News Category Classification"): This initializes MLflow. establishes the "Better-News Category Classification" MLflow experiment. All runs, parameters, and metrics will be recorded here. Define the function of experiment tracking:

Log Experiment Specifics: log_param: Logs the model's different parameters, including the kind of model and the number of estimators. These are crucial for monitoring the impact of model configurations on performance. print(accuracy, "Logging accuracy:"): prints the accuracy for confirmation prior to logging in.

log_metric("accuracy", accuracy): This logs the accuracy metric, one of the model's important performance metrics. Model: Save and log: mlflow.sklearn.log_model(model, "model") logs the model object in preparation for future deployment and versioning. The model is saved using this function in a format that MLflow can track and replicate at a later time. Carry Out the Tracking Task: The track_experiment() function is only called when the script is executed directly, not when it is imported as a module, thanks to the if **name** == "**main**": block.

Defines a method called track_experiment() that keeps track of and logs information about the model training procedure. utilizing mlflow.start_run(): Initiates an MLflow run, recording all metrics, models, and settings. Model Acquisition and Extraction: model, accuracy = train_models(): This calls the train_models function, which is thought to be used to train models and return the accuracy of the model object.

This code snippet integrates the use of MLflow, a popular platform for managing the machine learning lifecycle, including experimentation, reproducibility, and deployment. Here's a concise breakdown of what the script does:

1. **Import Statements**:
    - Imports `mlflow` and specific functions like `log_metric` and `log_param` for tracking experiment details.

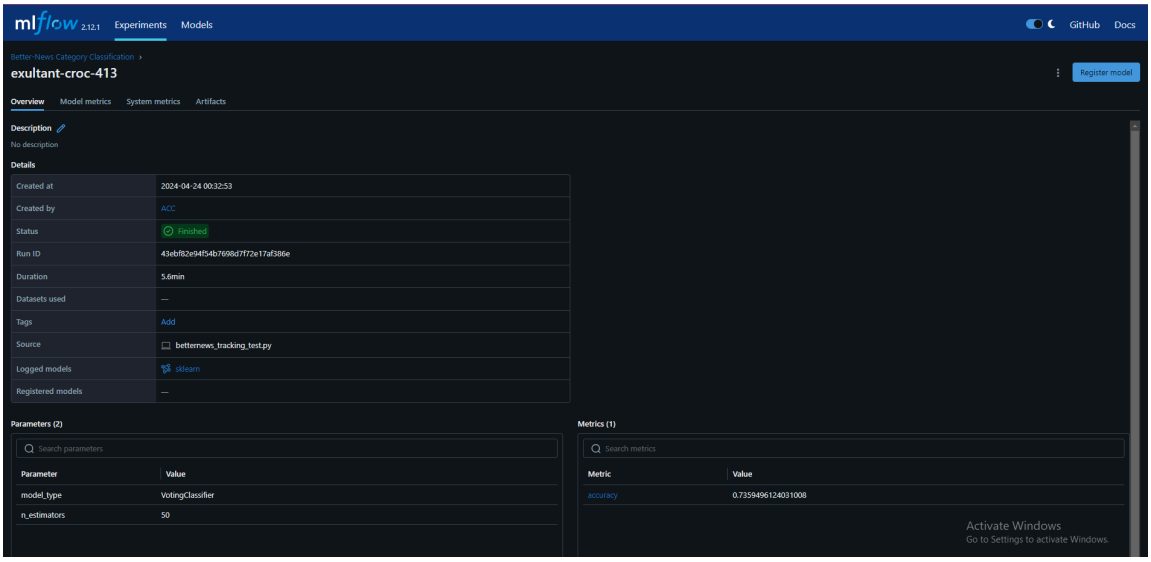
- Imports `train_models` from `betternews_testmodel`, which is assumed to be a script that trains models. Ensure this import is correctly pointing to the actual module where `train_models` is defined.

2. **Initialize MLflow**:
   - `mlflow.set_experiment("Better-News Category Classification")`: Sets up an MLflow experiment named "Better-News Category Classification". This is where all runs, parameters, and metrics will be tracked.
3. **Define Experiment Tracking Function**:
   - `track_experiment()`: Defines a function that tracks and logs details of the model training process.
   - `with mlflow.start_run()`: Starts an MLflow run, under which all parameters, metrics, and models are logged.
4. **Model Training and Retrieval**:
   - `model, accuracy = train_models()`: Calls the `train_models` function which presumably trains a model and returns the model object and its accuracy.
5. **Log Experiment Details**:
   - `log_param`: Logs various parameters of the model, such as the type of model and the number of estimators. These are important for tracking how model configurations affect performance.
   - `print("Logging accuracy:", accuracy)`: Prints the accuracy to verify it before logging.
   - `log_metric("accuracy", accuracy)`: Logs the accuracy metric, which is a key performance indicator for the model.
6. **Save and Log Model**:
   - `mlflow.sklearn.log_model(model, "model")`: Logs the model object for versioning and potential deployment. This function saves the model in a format that MLflow can track and reproduce later.
7. **Execute the Tracking Function**:
   - The `if __name__ == "__main__":` block ensures that `track_experiment()` is called only when the script is run directly (not when imported as a module).

At the end of this process, to view the logged experiments, parameters, and metrics, `mlflow ui` is run in the terminal. This command starts the MLflow dashboard, allowing you to interactively explore the experiments and their details via a web interface.

In [ ]:
```
import mlflow
from mlflow import log_metric, log_param
from betternews_testmodel import train_models  # Ensure this import is correct

# Initialize MLflow
mlflow.set_experiment("Better-News Category Classification")

def track_experiment():
    with mlflow.start_run():
        # Train models and retrieve model and accuracy
```

```
        model, accuracy = train_models()

        # Log parameters
        log_param("model_type", "VotingClassifier")
        log_param("n_estimators", 50)  # Example parameter

        # Log metrics
        print("Logging accuracy:", accuracy)  # Verify accuracy is being logged
        log_metric("accuracy", accuracy)  # Log accuracy as a metric

        # Save and log model
        mlflow.sklearn.log_model(model, "model")

# Execute the tracking function
if __name__ == "__main__":
    track_experiment()
```



Streamlit For UI Development

This Streamlit web application divides news articles into pre-established categories is defined by this code. It incorporates prediction, PDF text extraction, model loading, and text preparation. This is a brief explanaition of the codes below:

Libraries and Modules to be Imported: streamlit as st: Brings in the Streamlit library in order to construct the web application.

pickle: To load models that have been saved. BetterNews TestModel: Imports custom functions for handling model interactions, such as load_models_and_vectorizer and make_prediction. Fitz: Another name for PyMuPDF; a program for working with PDF files.

Load Models and Vectorizer: This function loads a text vectorizer together with the models by specifying the directory in which they are kept. To verify that loading was successful, a message is printed. Function for Extracting Text from PDFs:

extract_text_from_pdf(pdf_file): This function defines how to use PyMuPDF to extract text from an uploaded PDF file. It concatenates the text on each page as it iterates over the PDF.

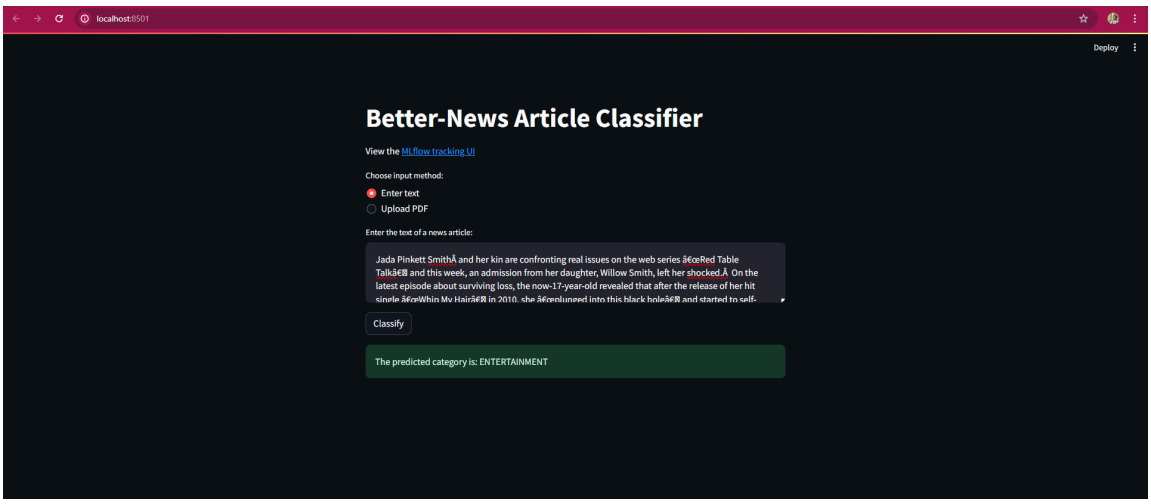
What is meant by a Streamlit Web Application? main(): The primary feature of the Streamlit app, which consists of: Title: Determines the web application's title.

Provides a clickable link to the MLflow tracking user interface. Input Method: Gives users the option to upload a PDF file or enter text directly. Text Input/Area: It offers a file uploader for PDFs or a text area for direct input, depending on the input type. Classification: When the "Classify" button is clicked, the input text is processed, the trained models are used to forecast the category, and the prediction is displayed. Launch the app Streamlit: The Streamlit application only launches when the script is invoked directly, thanks to the if **name** == "**main**": block. To launch the application, it invokes the main() function.

After configuring and saving the script, type streamlit start your_script_name.py in your terminal to launch and use the Streamlit application on your web browser. With this command, the program launches in your default web browser and the Streamlit server is started.

In [ ]:
```
import streamlit as st
import pickle
from betternews_testmodel import load_models_and_vectorizer, make_prediction, prepr
import fitz  # PyMuPDF

# Specify the directory name where the models are stored
models_dir = "models"

# Load the vectorizer and trained models
vectorizer, trained_models = load_models_and_vectorizer(models_dir)
print("Vectorizer and models loaded successfully.")

def extract_text_from_pdf(pdf_file):
    # Open the PDF from the uploaded file object
    doc = fitz.open(stream=pdf_file.read(), filetype="pdf")
    text = ""
    for page in doc:
        text += page.get_text()
    return text

# Streamlit app
def main():
    st.title("Better-News Article Classifier")

     # Link to MLflow UI
    st.markdown("View the [MLflow tracking UI](http://localhost:5000)", unsafe_allo


    # Toggle between text input and file upload
    input_option = st.radio("Choose input method:", ("Enter text", "Upload PDF"))

    if input_option == "Enter text":
        article_text = st.text_area("Enter the text of a news article:")
    elif input_option == "Upload PDF":
        pdf_file = st.file_uploader("Upload a PDF file", type=["pdf"])
        article_text = ""
        if pdf_file is not None:
            article_text = extract_text_from_pdf(pdf_file)

    if st.button("Classify"):
        if not article_text:
            st.warning("Please enter some text or upload a PDF.")
        else:
            # Preprocess the input text
            processed_article_text = preprocess_article_text(article_text)
            # Make prediction
```

```
            prediction = make_prediction(processed_article_text)
            st.success(f"The predicted category is: {prediction}")

if __name__ == "__main__":
    main()
```



Containerizing and Deploying with Docker and Heroku

In order to containerize the news classifying app and deploy it to the public Docker and Heroku were used to achieve this and here is how:

Make a Dockerfile. All the commands to put together an image on the command line are contained in a text document called a Dockerfile. Here is the News Classifier app's Dockerfile :

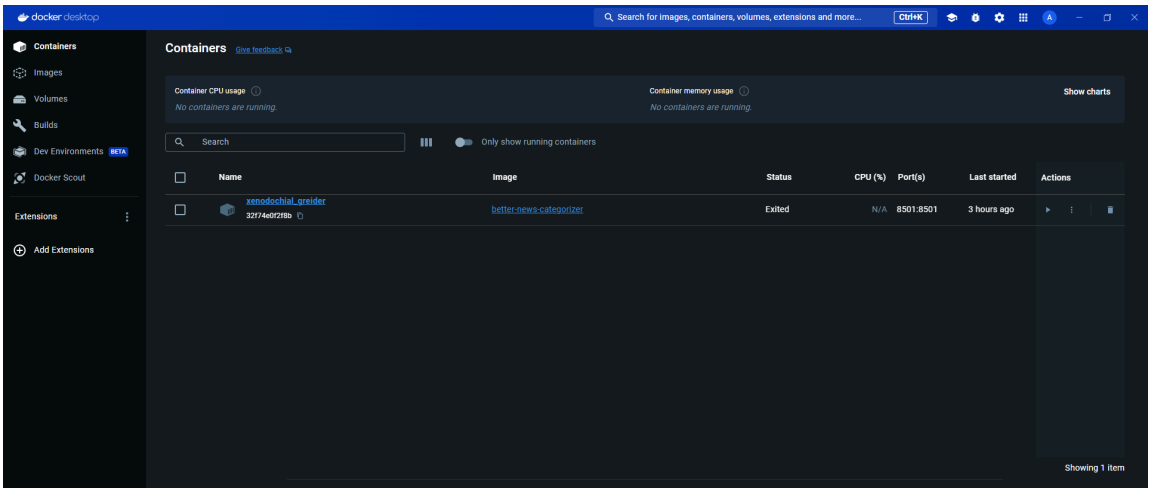
```
# Use an official Python runtime as a parent image
FROM python:3.12-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Make port 8501 available to the world outside this container
EXPOSE 8501

# Define environment variable
ENV NAME Better-News-Categorizer

# Run app.py when the container launches
CMD ["python", "betternews_test_app.py"]
```

Construct the Docker Image The Docker image is created after setting up the Dockerfile. In the same directory as the Dockerfile, the the follwing command is executed : build -t news-classification-app.

The Docker image is constructed and tagged as news-classification-app by this command.

A Local Docker Container Test is done before deploying with this command: Docker run -p 8501:8501 news-classification-app

By running the container and mapping port 8501 to port 8501 on the host machine to access the Streamlit app by opening a web browser and going to localhost:8501.



References

- Hayes, P. J., Knecht, L., & Cellio, M. J. (1988). A news story categorization system. The COCOON Platform (University of Paris). https://doi.org/10.3115/974235.974238
- Pyarasani, J. (2024, January 12). Building a Text Classification System for News Articles: A Comprehensive Guide. Medium. https://medium.com/@JyotsnaPyarasani/building-a-text-classification-system-for-news-articles-a-comprehensive-guide-10a99e8e862d
- Scikit-learn. (2019). scikit-learn: machine learning in Python — scikit-learn 0.20.3 documentation. Scikit-Learn.org. https://scikit-learn.org/stable/index.html

- Payne, R. (2023, September 22). 7 Best Python Libraries for Machine Learning and AI. Developer.com. https://www.developer.com/languages/python/python-libraries-for-machine-learning-ai/