Sentiment Analysis with Python

## Introduction

Today, we’ll explore an advanced Python script that performs news sentiment analysis and aggregation from RSS feeds. We’ll cover asynchronous programming, natural language processing, database operations, and task scheduling.

## Introduction to Key Concepts

### Asynchronous Programming

Asynchronous programming is a programming paradigm that allows multiple operations to be executed concurrently without blocking the main thread of execution. In Python, this is primarily achieved using the asyncio module and the async/await syntax.

Key benefits of asynchronous programming include:

1. Improved performance for I/O-bound tasks
2. Better resource utilization
3. Enhanced scalability for applications handling many concurrent operations

In our script, we use asynchronous programming to efficiently fetch data from multiple RSS feeds simultaneously, significantly reducing the overall execution time compared to a synchronous approach.

### Large Language Models (LLMs)

Large Language Models are advanced AI models trained on vast amounts of text data. They can understand and generate human-like text, and perform various natural language processing tasks such as sentiment analysis, text classification, and language translation.

Some key characteristics of LLMs include:

1. **Massive scale:** Trained on billions of parameters
2. **Versatility:** Can be fine-tuned for specific tasks
3. **Context understanding:** Can comprehend nuanced language use

In our script, we use a pre-trained BERT (Bidirectional Encoder Representations from Transformers) model, which is a type of LLM. Specifically, we’re using an Italian BERT model for sentiment analysis of Italian news articles.

We’re using an LLM for this task because:

1. **Accuracy:** LLMs can capture subtle nuances in language, leading to more accurate sentiment analysis.
2. **Efficiency:** Pre-trained models can be quickly adapted to our specific use case without extensive training.
3. **Multilingual capability:** We can easily switch to models for different languages if needed.

## Install

pip install asyncio aiohttp feedparser pandas transformers torch apscheduler

## Import Statements

import asyncio # framework for writing asynchronous code  
import aiohttp # an asynchronous HTTP client/server framework for asyncio  
import feedparser # for parsing RSS  
import csv  
import sqlite3 # working with SQLite databases  
from datetime import datetime  
from urllib.parse import urlparse #to parse URLs into their components  
import pandas as pd  
from transformers import AutoModelForSequenceClassification, AutoTokenizer  
# provides pre-trained models for natural language processing tasks  
import torch # used for machine learning tasks  
from apscheduler.schedulers.asyncio import AsyncIOScheduler  
# used for scheduling tasks asynchronously  
import logging

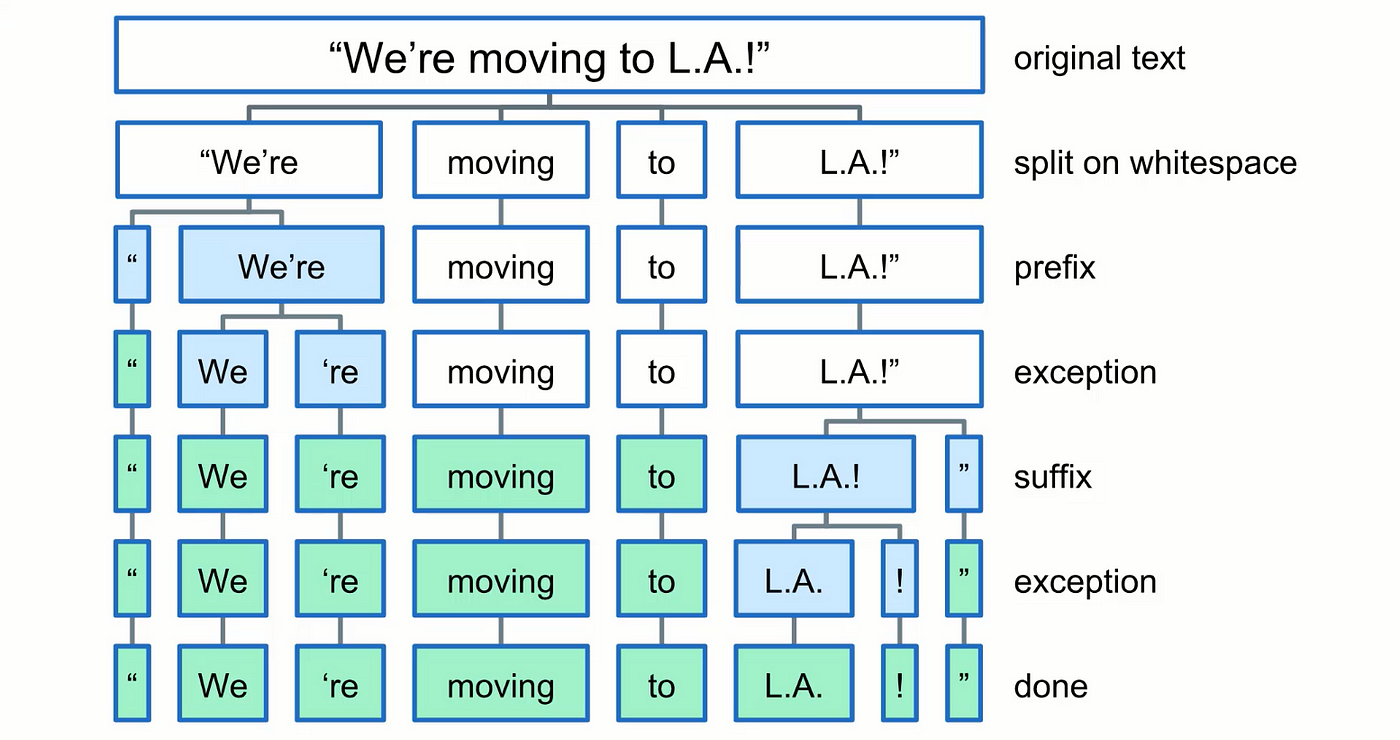
* AutoModelForSequenceClassification: Automatically loads a pre-trained model for sequence classification tasks.
* AutoTokenizer: Automatically loads the appropriate tokenizer for a given model.

# Set up logging  
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

## Model Initiliazation

# Initialize BERT model and tokenizer  
model\_name = "dbmdz/bert-base-italian-uncased"  
tokenizer = AutoTokenizer.from\_pretrained(model\_name)  
model = AutoModelForSequenceClassification.from\_pretrained(model\_name)

* model\_name: a pre-trained Italian BERT model.
* AutoTokenizer: Automatically loads the appropriate tokenizer for the model. Tokenizers convert text into a format the model can understand.
* AutoModelForSequenceClassification: Loads the pre-trained model for sentiment classification. This class automatically configures the model for sequence classification tasks.



We use these because they provide a convenient way to use NLP models without training from scratch.

## Data Sources and Keywords

RSS\_SOURCES = [  
 "https://www.repubblica.it/rss/homepage/rss2.0.xml",  
 "https://www.corriere.it/rss/homepage.xml",  
 # Now we will find RSS feeds  
]  
# You can adjust the keywords for the topics you wanna perform sentiment analysis.  
KEYWORDS = ["economia", "politica", "tecnologia", "salute", "nucleare", "covid"]

# This set will store processed article links to avoid duplicates.  
processed\_links = set()

## Sentiment Analysis Function

def analyze\_sentiment(text):  
 inputs = tokenizer(text, return\_tensors="pt", truncation=True, max\_length=512, padding=True)  
   
 with torch.no\_grad():  
 outputs = model(\*\*inputs)  
   
 logits = outputs.logits  
 probabilities = torch.softmax(logits, dim=1)  
 sentiment\_score = probabilities[0, 1].item() - probabilities[0, 0].item() # Positive - Negative  
   
 return sentiment\_score

* text: The input text to tokenize.
* return\_tensors="pt": Returns PyTorch tensors.
* truncation=True: Truncates text longer than the model’s maximum input length.
* max\_length=512: Sets the maximum length of the input.
* padding=True: Pads shorter sequences to the maximum length.

**Calculating Sentiment Score:**

sentiment\_score calculates the sentiment score by subtracting the negative sentiment probability from the positive sentiment probability.

## Keyword Search Function

def search\_keywords(text):  
 text\_lower = text.lower()  
 return [keyword for keyword in KEYWORDS if keyword in text\_lower]

comprehension returns all keywords found in the lowercase text

## Asynchronous News Fetching Function

async def fetch\_news\_async(rss\_url):  
 async with aiohttp.ClientSession() as session:  
 async with session.get(rss\_url) as response:  
 content = await response.text()  
 return feedparser.parse(content)

## Website Name Extraction Function

def get\_website\_name(url):  
 parsed\_url = urlparse(url)  
 return parsed\_url.netloc # returns the network location (domain) part of the URL

## Main Processing Function

async def process\_feeds():  
 results = [] # empty list to store the results  
 for source in RSS\_SOURCES:  
 feed = await fetch\_news\_async(source)  
 if feed:  
 for item in feed.entries:  
 link = item.get('link', '')  
 if link not in processed\_links: # checks if the link has already been processed  
 full\_text = item.get('title', '') + ' ' + item.get('summary', '')  
 sentiment = analyze\_sentiment(full\_text)  
 matched\_keywords = search\_keywords(full\_text)  
 results.append({  
 'title': item.get('title', ''),  
 'full\_text': full\_text,  
 'link': link,  
 'source': get\_website\_name(source),  
 'sentiment': sentiment,  
 'pub\_date': item.get('published', ''),  
 'author': item.get('author', ''),  
 'keywords': ', '.join(matched\_keywords)  
 })  
 processed\_links.add(link)  
 return results

## CSV Saving Function

def save\_to\_csv(data, filename='news\_sentiment.csv'):  
 with open(filename, 'a', newline='', encoding='utf-8') as file: # 'a' for append mode  
 writer = csv.DictWriter(file, fieldnames=data[0].keys())  
 if file.tell() == 0:  
 writer.writeheader()  
 writer.writerows(data)

## Database Saving Function

def save\_to\_database(data, db\_name='news\_sentiment.db'):  
 conn = sqlite3.connect(db\_name)  
 df = pd.DataFrame(data)  
 df.to\_sql('news', conn, if\_exists='append', index=False)  
 conn.close()

## Main Execution Function

async def main():  
 results = await process\_feeds()  
 if results:  
 save\_to\_csv(results)  
 save\_to\_database(results)  
 logging.info(f"Processed {len(results)} new news items.")  
 else:  
 logging.info("No new news items to process.")

## Scheduled Job Function

def scheduled\_job():  
 asyncio.run(main())

## Main Execution Block

if \_\_name\_\_ == "\_\_main\_\_":  
 # Run the job immediately  
 scheduled\_job()  
   
 # Set up the scheduler  
 scheduler = AsyncIOScheduler()  
 scheduler.add\_job(scheduled\_job, 'interval', hours=1) # you can change the hour  
 scheduler.start()  
 logging.info("Scheduler started. Press Ctrl+C to exit.")  
  
 try:  
 asyncio.get\_event\_loop().run\_forever()  
 except (KeyboardInterrupt, SystemExit):  
 pass



## To-Do

* Implement error handling and retries for RSS feed fetching
  + Add a retry mechanism for failed HTTP requests
  + Implement exponential backoff for repeated failures
* Enhance the sentiment analysis model
  + Consider fine-tuning the BERT model on domain-specific data
  + Implement a more nuanced sentiment scale (e.g., very negative, negative, neutral, positive, very positive)
* Improve topic modeling
  + Experiment with different numbers of topics for the NMF mode
  + Implement dynamic topic modeling to adapt to changing news trends
  + Consider using more advanced topic modeling techniques like LDA (Latent Dirichlet Allocation)
* Optimize database operations
  + Implement batch inserts for better performance
  + Add indexes to frequently queried columns
  + Consider using an ORM like SQLAlchemy for better database management
* Enhance the dashboard
  + Add more visualizations (e.g., word clouds, topic distribution pie charts)
  + Implement user authentication for the dashboard
  + Add the ability to export data in various formats (CSV, JSON, etc.)
* Implement caching
  + Use a caching mechanism (e.g., Redis) to store frequently accessed data and reduce database load
* Add more data sources
  + Integrate with news APIs for a broader range of sources
  + Implement web scraping for sources without RSS feeds
* Improve logging and monitoring
  + Implement more detailed logging for better debugging
  + Add system monitoring (e.g., CPU usage, memory usage, database performance)
* Enhance configurability
  + Make more parameters configurable through the config file
  + Implement a web interface for changing configuration settings
* Implement data cleaning and preprocessing
  + Add text cleaning functions (e.g., removing HTML tags, handling special characters)
  + Implement language detection to filter out non-Italian content
* Add unit tests and integration tests
  + Write unit tests for individual functions
  + Implement integration tests for the entire pipeline
* Optimize performance
  + Profile the code to identify bottlenecks
  + Implement multiprocessing for CPU-bound tasks
* Implement data retention policies
  + Add functionality to archive or delete old news items
  + Implement data compression for long-term storage
* Enhance specific topic detection
  + Implement machine learning-based classification for specific topics
  + Allow users to define and train custom topics
* Add trend detection
  + Implement algorithms to detect sudden changes in sentiment or topic popularity
  + Add alerts for significant trend changes