**RSS Feed News Categorization Application**

**1. Introduction**

This assignment details the development of an automated news article categorization system, which collects news articles from multiple RSS feeds, stores them in a relational database, and categorizes them into predefined categories using Natural Language Processing (NLP). The application is built using Python, SQLAlchemy, Celery, and NLP libraries like NLTK for tokenization and classification.

**Categories for Classification:**

* Terrorism / Protest / Political Unrest / Riot
* Positive / Uplifting
* Natural Disasters
* Others

**2. System Architecture Overview**

The application consists of three key components:

1. **RSS Feed Parsing**: Fetches news articles from RSS feeds and extracts relevant data.
2. **Database Management**: Stores parsed articles in a relational database.
3. **Asynchronous Processing and NLP**: Classifies articles into categories using NLP and updates the database.

**3. Detailed Implementation**

**3.1 RSS Feed Parsing and Data Extraction**

The first part of the system is responsible for fetching data from the provided RSS feeds. I used the **Feedparser** library to simplify the extraction of data from the RSS feeds, retrieving essential information like article title, content, publication date, and source URL.

**Key Steps:**

* A script is written to iterate through the list of RSS feeds (CNN, Fox News, Reuters, etc.).
* For each feed, articles are parsed using feedparser.parse(feed\_url).
* The relevant details (title, content, publication date, and source URL) are extracted and stored in the database.
* Duplicate detection is handled by comparing article URLs before saving.

Code Example:

def parse\_feed(feed\_url):

feed = feedparser.parse(feed\_url)

for entry in feed.entries:

title = entry.title

content = entry.description

publication\_date = entry.published

source\_url = entry.link

article = Article(title=title, content=content, publication\_date=publication\_date, source\_url=source\_url)

session.add(article)

session.commit()

celery.send\_task('process\_article', args=[article.id])

**3.2 Database Design and Storage**

The extracted articles are stored in a PostgreSQL database using **SQLAlchemy** ORM. The database schema consists of two main tables: articles and categories. An additional article\_categories table handles the many-to-many relationship between articles and their categories.

**Database Schema:**

* **Articles Table**: Stores information such as title, content, publication date, source URL, and assigned category.
* **Categories Table**: Contains the list of predefined categories.
* **Article\_Categories Table**: Manages the relationship between articles and categories.

**SQL Code Example:**

Sql code

CREATE TABLE articles (

id SERIAL PRIMARY KEY,

title VARCHAR(255) NOT NULL,

content TEXT NOT NULL,

publication\_date DATE NOT NULL,

source\_url VARCHAR(255) NOT NULL,

category VARCHAR(50) NOT NULL

);

CREATE TABLE categories (

id SERIAL PRIMARY KEY,

name VARCHAR(50) NOT NULL

);

CREATE TABLE article\_categories (

article\_id INTEGER NOT NULL,

category\_id INTEGER NOT NULL,

PRIMARY KEY (article\_id, category\_id),

FOREIGN KEY (article\_id) REFERENCES articles (id),

FOREIGN KEY (category\_id) REFERENCES categories (id)

);

**3.3 Task Queue and Asynchronous Processing**

Since we expect a continuous stream of news articles, I employed **Celery** to manage task queues and asynchronous processing. Celery allows the system to handle incoming articles efficiently without blocking other operations.

**Approach:**

* When an article is parsed and stored, a Celery task is triggered using celery.send\_task(), which sends the article to be processed asynchronously.
* A Celery worker processes each article by classifying it into the predefined categories based on the article's content.

**Celery Worker Example:**

Python code

celery = Celery('tasks', broker='amqp://guest@localhost//')

@celery.task

def process\_article(article\_id):

article = session.query(Article).get(article\_id)

# Process article content and classify

content = article.content

tokens = word\_tokenize(content)

tokens = [token for token in tokens if token.isalpha()]

tokens = [token for token in tokens if token not in stopwords.words('english')]

tokens = [WordNetLemmatizer().lemmatize(token) for token in tokens]

category = classify(tokens)

article.category = category

session.commit()

**3.4 Text Classification Using NLP**

The articles are categorized using a simple classification mechanism built with NLP techniques. I used **NLTK** for tokenizing and lemmatizing the article content and assigned a category based on pre-defined classes.

**Steps for Classification:**

* Tokenize the article content to extract words.
* Remove stopwords (common, uninformative words) and non-alphabetic tokens.
* Lemmatize the tokens to normalize words.
* Use a basic classifier to assign a category (this can be enhanced with machine learning models).

**Code Example for Classification:**

Python code

def classify(tokens):

# Simple random classification logic (for demonstration)

categories = ['Terrorism / protest / political unrest / riot', 'Positive/Uplifting', 'Natural Disasters', 'Others']

return random.choice(categories)

**3.5 Logging and Error Handling**

Logging is implemented throughout the application to ensure smooth operation and error tracking. The **logging** module in Python is used to capture important events, such as successful parsing of an article or errors in network communication.

**Error Handling:**

* Network issues during feed fetching are logged, and retries are handled gracefully.
* Parsing errors (e.g., malformed feeds) are logged for future reference.

**Example:**

Python code

import logging

logging.basicConfig(level=logging.INFO)

def parse\_feed(feed\_url):

try:

feed = feedparser.parse(feed\_url)

logging.info(f"Successfully parsed feed: {feed\_url}")

except Exception as e:

logging.error(f"Error parsing feed: {feed\_url}, Error: {e}")

**4. Technologies and Libraries Used**

* **Python**: Main programming language.
* **Feedparser**: For extracting data from RSS feeds.
* **SQLAlchemy**: For database interaction (PostgreSQL).
* **Celery**: For task management and asynchronous processing.
* **NLTK**: For natural language processing (tokenization, lemmatization, etc.).
* **PostgreSQL**: Relational database for storing articles.
* **Flask**: (Optional) For building a simple API interface.

**5. Conclusion**

This document presents a systematic approach to building an application that collects and processes news articles from RSS feeds, storing them in a relational database and categorizing them using basic NLP techniques. By utilizing tools like **Feedparser**, **Celery**, and **SQLAlchemy**, the system efficiently handles real-time news data and categorizes it into meaningful categories.

**6. Future Improvements**

* Implement more sophisticated classification models (e.g., machine learning classifiers).
* Add support for real-time news feed updates using webhooks or data streaming.
* Incorporate additional error handling mechanisms for more robust performance.