Documentation for News Article Collector Application

Overview

This application is designed to collect news articles from various RSS feeds, categorize them into predefined categories, and store them in a relational database. The application employs Python libraries to handle feed parsing, database interactions, and natural language processing for categorization.

Categories

The application classifies articles into the following categories:

- Terrorism / protest / political unrest / riot
- Positive/Uplifting
- Natural Disasters
- Others

Requirements

- **Programming Language:** Python
- **Libraries:**
    - feedparser for parsing RSS feeds
    - SQLAlchemy for database interactions
    - pandas for data manipulation
    - BeautifulSoup for HTML content cleaning
    - nlp_utilities (custom) for article classification
- **Database:** MySQL (or any relational database)

Setup Instructions

1. **Install Required Libraries:** Install the necessary libraries using pip:

   → pip install feedparser sqlalchemy pandas beautifulsoup4 mysql-connector-python

2**. Database Configuration:**

- Create a MySQL database to store news articles.
- Update the config.yaml file with your database credentials.

3. **Create Configuration File (config.yaml):**

```
Database:
  user: your_db_user
  password: your_db_password
  name: your_db_name
```

4. **Run the Application:** Execute the main.py file:

Code Structure

1. Imports

The script imports necessary libraries for logging, RSS feed parsing, database interaction, and HTML content handling.

```
import logging
import feedparser
import configparser
import pandas as pd
from datetime import datetime
from bs4 import BeautifulSoup
from sqlalchemy import create_engine, Column, String, Text, DateTime, exc
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from nlp_utilities import classify_article
```

2. Logging Configuration

The logging system is set up to log events and errors to a file and the console.

```
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler("app.log"),
        logging.StreamHandler()
    ]
)
```

3. Main Function

The main function orchestrates the application's workflow:

- Reads database credentials from the config.yaml file.
- Initializes the database connection and defines the NewsArticle model.
- Creates the database table if it doesn't exist.
- Parses RSS feeds and stores the articles in the database.

4. Database Model

The NewsArticle class defines the schema for storing news articles in the database.

```
class NewsArticle(Base):
    __tablename__ = 'news_articles'

    id = Column(String(255), primary_key=True) # Consider Link as Primary Key
    title = Column(String(255))
    link = Column(String(255), unique=True)
    description = Column(Text)
    published_date = Column(DateTime)
    category = Column(String(255))
```

5. Feed Parsing

The parse_feeds function reads each RSS feed, extracts relevant information, and stores articles in the database.

6. Data Cleaning and Formatting

The application includes utility functions to clean HTML content and format publication dates.

7. Duplicate Handling

Before storing an article, the application checks for existing entries to avoid duplicates based on the unique article link.

8. Error Handling

The application implements logging to track errors during feed parsing and database operations.

Articles Classification

Overview

The nlp_utilities.py module is designed for classifying text data, such as news articles, into specific categories based on keyword matching, n-gram models, and normalized scoring. The primary categories include:

- Terrorism
- Protest
- Natural Disasters
- Positive/Uplifting

The classification is performed by matching pre-defined sets of keywords, bi-grams, and tri-grams related to each category, followed by normalized scoring. The module also supports text preprocessing by removing stop words, HTML tags, punctuation, and numbers.

Dependencies

- **Python Libraries:**
  - re: For regular expressions and text manipulation.
  - nltk: For natural language processing, tokenization, and handling n-grams.
  - nltk.corpus.stopwords: To remove common English stop words from the text.

File Structure

The script uses several files to load keywords for different categories, and they should be organized in a Corpus directory:

- Corpus/natural disaster terms.txt
- Corpus/natural disaster unique terms.txt
- Corpus/positive terms.txt
- Corpus/positive unique terms.txt

- Corpus/protest terms.txt
- Corpus/protest unique terms.txt
- Corpus/Terrorism terms.txt
- Corpus/Terrorism unique terms.txt

These files contain keywords (one per line) that are used for keyword matching in classification.

Functions

1. load_keywords(file_path)

- **Description**: This function reads keywords from a specified file, converts them to lowercase, and returns a set of terms for efficient keyword lookup.
- **Input**:
  - file_path (str): The path to the file containing the keywords.
- **Output**:
  - Returns a set of keywords in lowercase.

2. preprocess_text(text)

- **Description**: Cleans and tokenizes input text by removing HTML tags, punctuation, and numbers. Converts the text to lowercase and removes common English stop words.
- **Input**:
  - text (str): The input text to preprocess.
- **Output**:
  - Returns a list of tokens (words) after preprocessing.
- **Notes**: If the input is not a string, it returns an empty list.

3. get_ngrams(tokens, n=2)

- **Description**: Generates n-grams from a list of tokens. By default, it generates bi-grams (pairs of adjacent words), but the n parameter can be adjusted to create tri-grams, etc.
- **Input**:
  - tokens (list): A list of preprocessed tokens.
  - n (int): The number of tokens in each n-gram (default is 2).
- **Output**:
  - Returns a list of n-grams (tuples of n adjacent words).

4. classify_article(text, threshold=0.0003)

- **Description**: Classifies the input text into one of the following categories: "Terrorism", "Protest", "Natural Disasters", "Positive/Uplifting", or "Others". The classification is based on keyword matches and n-gram matches from pre-defined keyword sets.
- **Process**:
  - **Preprocessing**: The text is preprocessed to remove stop words, punctuation, and unnecessary characters.
  - **Keyword Matching**: Counts the occurrences of keywords and n-grams (bi-grams and tri-grams) from each category in the text.
  - **Score Normalization**: The raw scores for each category are normalized by dividing by the total number of keywords for that category.
  - **Classification Decision**: The text is classified into the category with the highest normalized score, provided the score is above the threshold.
- **Input**:

    - text (str): The text to classify.
    - threshold (float): The minimum normalized score required to classify the text into a category (default is 0.0003).
- **Output**:
    - Returns a string representing the classified category, or "Others" if no category meets the threshold.