National University of Computer and Emerging Sciences
Islamabad Campus

Assignment 02

MLOps Implementation with Apache Airflow

MLOPs

Submitted by: Alina Aftab
Roll number: 20i-0961
Date: 12/05/2024

Table of Content

National University of Computer and Emerging Sciences
Islamabad Campus

Introduction

The objective of this project was to implement Apache Airflow to automate the processes of data extraction, transformation, and version-controlled storage from two news websites: dawn.com and BBC.com. The key tasks included extracting links, titles, and descriptions from the homepage of each website, preprocessing the extracted text data, storing the processed data on Google Drive, and implementing Data Version Control (DVC) to track versions of the data.

Implementation

Data Extraction

Utilizing the BeautifulSoup library in Python, the data extraction process involved retrieving information from the landing pages of dawn.com and BBC.com. Here's a detailed breakdown of the extraction steps:

Extracting Links:

1. The BeautifulSoup library was employed to parse the HTML structure of the landing pages.
2. Specific HTML elements such as <a> tags containing links were targeted using CSS selectors or XPath expressions.
3. The extracted links were then stored for further processing or analysis.

Extracting Titles and Descriptions:

1. Similar to link extraction, BeautifulSoup was utilized to navigate through the HTML structure of article sections on the homepages.
2. Title and description elements were identified using CSS selectors or XPath expressions.
3. Text content within these elements was extracted, capturing the titles and descriptions of articles displayed on the homepages.

Data Transformation

After extracting text data from the websites, it underwent preprocessing to ensure

cleanliness and suitability for analysis. The transformation steps included:

Cleaning Text Data:

1. Removal of HTML tags, if any, using libraries like BeautifulSoup or regular expressions.
2. Stripping special characters, punctuation, and non-alphanumeric symbols from the text.
3. Converting text to lowercase to standardize case usage.

Tokenization and Lemmatization:

1. Tokenization involved breaking down the text into individual words or tokens.
2. Lemmatization was performed to reduce words to their base or root form, aiding in normalization and reducing vocabulary size.
3. Stopwords, common words like "the," "and," etc., were removed to focus on meaningful content.

Data Storage and Version Control

The processed data was stored on Google Drive using the Google Drive API, ensuring accessibility and reliability. Additionally, Data Version Control (DVC) was implemented to track versions of the data and metadata. Here's how it was done:

Storing Data on Google Drive:

1. The Google Drive API was utilized to programmatically upload and organize the processed data on Google Drive.
2. Authentication mechanisms were implemented to securely access the Google Drive account.

Implementing Data Version Control (DVC):

1. DVC was integrated with the project to track changes made to the data over time.
2. Each version of the data was recorded and stored separately, facilitating easy rollback and comparison.
3. Metadata related to each data version was versioned alongside the data, ensuring accurate recording of changes.

4. Integration with GitHub allowed for collaboration and sharing of the version-controlled data.

Apache Airflow DAG Development

Apache Airflow was leveraged to automate the entire data pipeline, encompassing extraction, transformation, and storage processes. Key aspects of DAG development included:

Automating Extraction, Transformation, and Storage:

1. A DAG (Directed Acyclic Graph) was developed to orchestrate the sequential execution of tasks.
2. Tasks for data extraction, transformation, and storage were defined within the DAG, each representing a distinct step in the pipeline.
3. Dependencies between tasks were established to ensure proper order of execution.

Error Management and Notifications:

1. Error handling mechanisms were implemented within the DAG to handle exceptions and failures gracefully.
2. Retries were configured for tasks that might encounter transient errors, ensuring robustness of the workflow.
3. Email notifications were set up to alert stakeholders in case of task failures or abnormalities, enabling timely intervention.

Overall, the Apache Airflow DAG provided a scalable and efficient solution for automating data workflows, streamlining processes, and enhancing productivity.

Challenges Encountered

Data Extraction

Parsing dynamic content from web pages can be challenging due to the asynchronous

loading of elements via JavaScript. To address this, a comprehensive understanding of the HTML structure and the functioning of JavaScript-rendered elements was crucial.

HTML Structure Inspection:

1. Utilizing tools like browser developer tools (e.g., Chrome DevTools) helped in inspecting the HTML structure of the web pages.
2. Identifying the specific elements containing the desired content, such as article titles, descriptions, and links, required careful examination.
3. Dynamic content loaded via JavaScript was observed to determine its placement and behavior within the HTML DOM.

Handling JavaScript-Rendered Elements:

1. BeautifulSoup, while powerful for static HTML parsing, may not always capture dynamically loaded content.
2. Solutions like Selenium WebDriver were employed to interact with the web page in a browser-like environment, enabling the execution of JavaScript and retrieval of dynamically generated content.
3. Selenium scripts were written to simulate user interactions and capture the desired data post-rendering.

Data Transformation

Designing an effective preprocessing pipeline involves striking a balance between thorough data cleaning and preserving the meaningful information present in the text. Challenges arise particularly when dealing with diverse text formats from different sources.

Data Cleaning:

1. Removal of HTML tags and special characters was performed using BeautifulSoup or regular expressions.
2. Standardizing text by converting to lowercase and removing non-alphanumeric characters ensured consistency.
3. Techniques such as stemming and lemmatization were applied to reduce inflected words to their base form, reducing vocabulary size and enhancing

analysis efficiency.

Preservation of Meaningful Information:

1. Retaining contextually relevant information while filtering out noise and irrelevant text required careful consideration.
2. Strategies like entity recognition and named entity recognition (NER) were employed to identify and preserve entities, such as proper nouns and key phrases, which carry significance.
3. Tokenization techniques were applied to break down text into individual tokens, facilitating further analysis and feature extraction.

DVC Integration

Configuring Data Version Control (DVC) to seamlessly integrate with Google Drive and GitHub posed challenges, necessitating a thorough understanding of DVC principles and integration steps.

Google Drive Integration:

1. Leveraging the Google Drive API, authentication mechanisms were implemented to securely access the Google Drive account.
2. Data storage on Google Drive required careful organization and structuring to ensure easy access and retrieval.

GitHub Integration:

1. Integration with GitHub facilitated collaborative version control, enabling multiple users to contribute to the project.
2. Metadata related to each data version, such as commit messages and timestamps, were accurately recorded and versioned alongside the data.
3. Configuring DVC pipelines to interact with both Google Drive and GitHub repositories required meticulous setup and configuration.

Version Tracking

1. Ensuring accurate version tracking across Google Drive and GitHub repositories necessitated consistent naming conventions and synchronization between DVC and version control systems.
2. Regular audits and reviews of version histories were conducted to validate the integrity and completeness of versioned data and metadata.

Addressing the challenges in data extraction, transformation, and DVC integration required a combination of technical expertise, meticulous planning, and iterative refinement of workflows. Overcoming these challenges laid the foundation for a robust and scalable data pipeline capable of extracting, transforming, and version-controlling data effectively.

Conclusion

The implementation of Apache Airflow for automating data extraction, transformation, and version-controlled storage proved successful in achieving the project objectives. The workflow streamlined the data pipeline, improving efficiency and repeatability while ensuring version control and traceability of changes made to the data.

GitHub Repository

https://github.com/alinaftabb/mlopsa2

Documentation

Data Preprocessing Steps:

Text Extraction: Utilizing BeautifulSoup library, the HTML content of the news articles' titles and descriptions was parsed to extract text data.

Text Cleaning:

1. Removal of HTML tags: HTML tags such as `<p>`, `<div>`, etc., were removed using regex patterns to clean the text.
2. Special character removal: Special characters like punctuation marks, symbols, and non-alphanumeric characters were removed to ensure only meaningful text remained.
3. Lowercasing: All text was converted to lowercase to ensure consistency and facilitate further analysis.

Tokenization and Lemmatization:

1. Tokenization: Text was tokenized into individual words or tokens using the nltk library's word_tokenize function.
2. Lemmatization: Each token was lemmatized using WordNetLemmatizer from nltk to convert words to their base or dictionary form, reducing inflectional forms to a common base.

Stopword Removal: Stopwords, common words like 'and', 'the', 'is', etc., were removed using nltk's stopwords corpus to reduce noise in the text data.

Data Formatting: The cleaned and processed text data was formatted into appropriate data structures such as lists or pandas DataFrames for further analysis and storage.

DVC Setup:

Installation: DVC was installed via pip using the command `pip install dvc`.

Initialization: DVC was initialized in the project directory using the command `dvc init`.

Data Tracking: Data files were added to DVC using the command `dvc add <file>` to start tracking changes.

Google Drive Integration: Google Drive was integrated as a remote storage backend for DVC using the command `dvc remote add -d myremote gdrive://<folder_id>` where `<folder_id>` is the ID of the Google Drive folder.

GitHub Integration:

1. DVC was integrated with GitHub for version control of metadata using the

command `dvc remote modify myremote --local --local -t gdrive` to specify the remote storage type.
2. Metadata was versioned against each DVC push to the GitHub repository, ensuring accurate recording of changes made to the data.

Pushing Changes: Changes to data files were committed using `git add` and `git commit`, followed by `dvc push` to push both data and metadata changes to remote storage.

Pulling Changes: Changes made by other team members were pulled using `git pull` followed by `dvc pull` to update local data files and metadata accordingly.

By following these steps, we established a robust data preprocessing pipeline and implemented effective version control using DVC, ensuring reproducibility and traceability of changes in the data.

Future Improvements

1. Integration of additional data sources for broader coverage.
2. Implementation of advanced text processing techniques for better data quality.
3. Enhancements to error handling and fault tolerance mechanisms in the Airflow DAG.

Overall, this project demonstrates the power and flexibility of Apache Airflow in orchestrating complex data workflows and lays the foundation for further automation and optimization in data processing tasks.

ScreenShots

National University of Computer and Emerging Sciences
Islamabad Campus