INTERNSHIP PROJECT

PRESENTED BY

ANAIS ANAND KALARICKAL

INDEX

TASK 1

Sentiment analysis, a natural language processing technique, gauges emotional tone in text. It assesses whether the expressed sentiment is positive, negative, or neutral. By analyzing words, phrases, and context, it aids in understanding public opinion, customer feedback, and social media sentiment. This helps businesses and researchers make data-driven decisions and track reactions to products, services, or events, enhancing customer satisfaction and market insights.

MODELS USED

VADER Model:

VADER is a rule-based model that uses a pre-built lexicon of words and their associated sentiment scores.

It analyzes text to determine sentiment (positive, negative, or neutral) and provides a sentiment score.

RoBERTa Model:

RoBERTa is a deep learning model based on the Transformer architecture, which excels at understanding context in text.

It's pre-trained on a massive amount of text data and fine-tuned for specific NLP tasks, including sentiment analysis.

WHY IS ROBERTA BETTER THAN VADER?

Contextual Understanding

Adaptability

Accuracy

Handling Ambiguity

Longer Texts

TASK 3

DATASET: TITANIC.CSV

The sinking of the Titanic is one of the most infamous shipwrecks in history.

On April 15, 1912, during her maiden voyage, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren’t enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew.

While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others.

In this challenge, we ask you to build a predictive model that answers the question: “what sorts of people were more likely to survive?” using passenger data (ie name, age, gender, socio-economic class, etc).

FEATURES OF THE DATASET

survival - Survival 0 = No, 1 = Yes

pclass - Ticket class 1 = 1st, 2 = 2nd, 3 = 3rd

sex - Sex

Age - Age in years

sibsp - # of siblings / spouses aboard the Titanic

parch - # of parents / children aboard the Titanic

ticket - Ticket number

fare - Passenger fare

cabin - Cabin number

embarked - Port of Embarkation C = Cherbourg, Q = Queenstown, S = Southampton

STEPS

1. Download the dataset: Dataset

2. Load the dataset into the tool.

3. Perform Below Visualizations.

● Univariate Analysis

● Bi- Variate Analysis

● Multi-Variate Analysis

4. Perform descriptive statistics on the dataset.

5. Check for Missing values and deal with them.

6. Find the outliers and replace them outliers

7.Check the correlation of independent variables with the target

8. Check for Categorical columns and perform encoding.

9. Split the data into dependent and independent variables.

10. Scaling the data

11. Split the data into training and testing

12.check the training and testing data shape

13. GridSearch CV to optimize the model

14. Implementing the classification model

15. Checking the accuracy

Since its a Classification question,

Did HyperParameter tuning using Grid Search CV

and optimised the dataset and then iimplemented various models such as :

LOGISTIC REGRESSION

SVC

DECISION TREE CLASSIFIER

TASK 2

RESTful API

Introduction

This project is a simple Flask-based RESTful API for managing a collection of books. The API allows you to perform basic CRUD (Create, Read, Update, Delete) operations on a collection of books.

Technologies Used

- **Flask**: A micro web framework for Python used to build the API.
- **VS Code**: An integrated development environment (IDE) used for coding, testing, and running the Flask application.
- **Insomnia**: An API testing tool used to send HTTP requests to the Flask API and receive responses.

Role of VS Code

VS Code, the integrated development environment, played a crucial role in the development of this project. Here's what was accomplished using VS Code:

- **Coding and Development**: VS Code was used as the primary code editor to write the Flask API code. It allowed for efficient code development with features such as code highlighting, auto-completion, and integrated terminal support.
- **Testing**: The VS Code terminal was used to run the Flask application, enabling the testing of the API locally. It was essential for ensuring that the API worked as expected.
- **Debugging**: VS Code provides debugging capabilities, which were useful for identifying and resolving any issues or errors in the Flask code.
- **Version Control**: VS Code can be integrated with version control systems (e.g., Git). It was used to manage the project's source code, making it easier to collaborate and track changes.

Role of Insomnia

Insomnia, the API testing tool, was used to interact with the Flask API, send requests, and receive responses. Here's what was accomplished using Insomnia:

- **API Testing**: Insomnia was used to send various HTTP requests (GET, POST, PUT, DELETE) to the Flask API. It allowed for thorough testing of the API's functionality.
- **Validation**: Insomnia was used to validate the correctness of API responses, ensuring that the Flask API was returning the expected data.

- **Documentation Testing**: Insomnia helped in documenting how to use the API effectively. It served as a testing tool to verify the accuracy of the documented API endpoints and expected responses.

Flask API Code

The Flask API code is divided into the following components:

- **GET Endpoint**: **`/api/books`** (GET) - This endpoint returns a list of books.
- **GET Endpoint**: **`/api/books/<book_id>`** (GET) - This endpoint retrieves a specific book by its ID.
- **POST Endpoint**: **`/api/books`** (POST) - This endpoint is used to create a new book.
- **PUT Endpoint**: **`/api/books/<book_id>`** (PUT) - This endpoint is used to update the information of a specific book.
- **DELETE Endpoint**: **`/api/books/<book_id>`** (DELETE) - This endpoint is used to delete a book by its ID.

The API uses a list of dictionaries to simulate a database, containing book information.

Conclusion

This documentation outlines the roles of VS Code and Insomnia in the development and testing of the Flask API. It also provides a brief overview of the Flask API's endpoints and functionality, which can be used as a reference for working with the API.

This project demonstrates how the combination of development tools and testing tools can be used effectively in the creation and testing of web APIs.

GraphQL

Introduction

This project presents a Flask-based GraphQL API for managing a collection of books. The API provides functionalities for creating, reading, updating, and deleting books. This documentation outlines the roles of development tools and the structure of the Flask GraphQL API.

Technologies Used

- **Flask**: A micro web framework used for building the API.
- **VS Code**: An integrated development environment (IDE) used for coding, debugging, and running the Flask application.
- **Insomnia**: An API testing tool used to send GraphQL queries and mutations to the Flask GraphQL API and validate responses.

Role of VS Code

VS Code, the integrated development environment, played a critical role in developing and testing the Flask GraphQL API. Here's how VS Code was utilized:

- **Coding and Development**: VS Code served as the primary code editor for writing the Flask GraphQL API code. It provided essential features like code highlighting, auto-completion, and an integrated terminal.
- **Testing and Debugging**: The VS Code terminal was used for running the Flask application locally, allowing testing and debugging of the API. It helped identify and resolve issues in the code.
- **Version Control**: VS Code was integrated with version control systems (e.g., Git) to manage the project's source code. This facilitated collaboration and version tracking.

Role of Insomnia

Insomnia, the API testing tool, played a vital role in interacting with the Flask GraphQL API, sending GraphQL requests, and receiving responses. Here's how Insomnia was used:

- **API Testing**: Insomnia was used to send GraphQL queries and mutations to the Flask GraphQL API. It enabled thorough testing of the API's functionality.
- **Validation**: Insomnia was employed to validate the correctness of API responses, ensuring that the Flask GraphQL API returned the expected data.

Flask GraphQL API Code

The Flask GraphQL API code is structured as follows:

- **GraphQL Schema**: The GraphQL schema defines the data types, queries, mutations, and resolvers. It includes a "Book" type and defines queries for listing books and retrieving a single book. Mutations for creating, updating, and deleting books are also specified.
- **Sample Data**: A list of dictionaries is used to simulate a database, containing book information. This sample data is provided for testing and demonstration purposes.
- **Query and Mutation Resolvers**: Resolvers are defined to fetch and manipulate data based on GraphQL queries and mutations. Resolvers connect to the sample data and return responses.
- **Integration with Flask**: The Flask application is created and configured to run the GraphQL API using the `Flask-GraphQL` extension. It also enables the GraphiQL interface for easy API exploration.

Conclusion

This documentation summarizes the roles of VS Code and Insomnia in the development and testing of the Flask GraphQL API. It also provides an overview of the API's structure, including data types, queries, mutations, and resolvers. This project demonstrates the effective use of development and testing tools in creating web APIs.