A
PROJECT DOCUMENTATION
ON

FRAUD DETECTION SYSTEM

SUBMITTED BY
Amit Duwal

SUBMITTED TO
GENESE Solutions, Nepal

May 18, 2023

Introduction

A fraud detection system is a technology-driven solution designed to identify and prevent fraudulent activities or transactions. It employs various techniques, such as data analysis, machine learning, and pattern recognition, to detect suspicious behavior and potential fraud in real-time or post-transaction analysis.

Here are some key components and functionalities typically found in a fraud detection system:

1. Data Collection: The system collects relevant data from various sources, including transaction records, user profiles, device information, and historical data. This data serves as the basis for analyzing and identifying fraudulent patterns.

2. Data Analysis: Advanced analytics techniques are applied to the collected data to uncover patterns, anomalies, and correlations that may indicate fraudulent activity. Machine learning algorithms are often used to learn from historical data and adapt to new fraud patterns.

3. Anomaly Detection: By establishing baselines and understanding normal behavior, the system can detect anomalies or deviations from expected patterns. Unusual transaction amounts, uncommon transaction locations, or atypical user behavior can trigger alerts for potential fraud.

4. Real-Time Monitoring: Many fraud detection systems operate in real-time, continuously monitoring transactions as they occur. This enables immediate detection and response to potential fraudulent activities, minimizing the impact of fraud on businesses and customers.

5. Integration with External Data Sources: Fraud detection systems may integrate with external data sources, such as blacklists, watchlists, or public databases, to enrich the analysis and improve the accuracy of fraud detection. These sources provide additional information about known fraudsters, compromised accounts, or suspicious entities.

6. Case Management and Investigation: When suspicious activity is detected, the system generates alerts or cases for further investigation. Fraud analysts or investigators can review the flagged cases, gather additional evidence, and make informed decisions regarding the legitimacy of the activity.

Fraud detection systems are utilized in various industries, including banking, e-commerce, insurance, healthcare, and telecommunications, where fraud poses a significant risk. The goal is to protect businesses, customers, and stakeholders from financial losses, reputational damage, and operational disruptions caused by fraudulent activities.

Requirements

To run the Stock Price Checker, you will need the following:

- Python 3.6 or later
- Flask
- Prefect
- Pandas
- Scikit-Learn

Features

Real-Time Data Generation:

The project includes the generation of real-time transaction data in the form of two CSV files. These files simulate the continuous flow of transactions that occur in real-world scenarios. The generation of real-time data allows for testing and validating the fraud detection system's ability to identify fraudulent activities promptly.

Data Processing:

The fraud detection system processes the incoming data to identify potential fraudulent transactions. The processing involves applying preprocessing techniques, feature engineering, and feature scaling to prepare the data for the trained machine learning model. Preprocessing steps may include data cleaning, handling missing values, encoding categorical features, and normalization.

Fraud Detection Model:

The project employs a machine learning model, such as logistic regression, to detect fraudulent transactions. The model is trained using historical data and evaluated on a separate testing dataset. The trained model can then predict whether a new transaction is fraudulent or legitimate based on the input features.

Handling Imbalanced Data:

Imbalanced datasets, where the number of fraudulent transactions is significantly lower than legitimate transactions, pose a challenge for fraud detection. To address this issue, the project utilizes techniques like SMOTE (Synthetic Minority Over-sampling Technique) to balance the dataset. SMOTE generates synthetic samples for the minority class (fraudulent transactions), improving the model's ability to detect fraud accurately.

Real-Time Data Integration with Prefect:

The project integrates the real-time transaction data with Prefect, a workflow management system. Prefect allows for the creation of data processing pipelines, scheduling of tasks, and monitoring of data flows. Integration with Prefect ensures the efficient consumption of real-time data by the fraud detection system and enables better management and orchestration of the overall workflow.

Web Application for Monitoring:

A web application is developed using Flask, a Python web framework, to monitor transactions and provide an interface for viewing details of fraudulent activities. The web application allows users to visualize transaction histories, track suspicious patterns, and access additional information related to fraudulent transactions. It provides a user-friendly and interactive platform for monitoring and analyzing detected fraudulent activities.

Scalability and Performance:

The fraud detection system is designed to handle large volumes of data efficiently and scale as the transaction volume increases. This includes optimizing data processing algorithms, leveraging parallel processing techniques, and utilizing distributed computing frameworks if necessary.

Monitoring and Reporting:

The project includes monitoring and reporting capabilities to track the performance of the fraud detection system. Key performance metrics such as detection accuracy, false positive rate, precision, recall, and F1-score can be monitored over time. Additionally, comprehensive reports can be generated to

provide insights into fraud patterns, system performance, and potential areas for improvement.

Technologies used

Python:

Python is a high-level programming language that is widely used for web development, data science, artificial intelligence, and other applications. Python 3.6 is a version of Python that was released in December 2016. It introduced many new features, including f-strings for string interpolation, async and await keywords for asynchronous programming, and improved handling of dictionaries.

Flask:

Flask is a micro web framework written in Python. It is designed to be lightweight and flexible, making it a popular choice for building web applications. Flask provides a simple and intuitive interface for handling HTTP requests and responses, and supports a wide range of extensions for adding functionality such as database integration, authentication, and more.

Prefect:

Prefect is a workflow management system that simplifies the creation, scheduling, and monitoring of data processing pipelines. It provides a Python-based infrastructure for defining and executing workflows, making it easier to orchestrate complex data tasks. Prefect allows developers to design workflows as code, ensuring repeatability and reproducibility in data processing pipelines. It offers features such as task dependencies, automatic retries, and error handling, ensuring robustness and fault tolerance. Prefect integrates well with various data processing libraries and frameworks, making it a valuable tool for managing the flow of data in the fraud detection project.

Pandas:

Pandas is a popular open-source Python library for data manipulation and analysis. It provides data structures and functions for efficiently handling structured data, such as CSV files, Excel spreadsheets, and databases. Pandas' DataFrame is a key data structure used in the fraud detection project, enabling easy manipulation, filtering, and transformation of the transaction data. With Pandas, you can perform operations like filtering rows based on specific conditions, aggregating data, handling missing values, and joining multiple

datasets. It also supports data cleaning, feature engineering, and feature selection, which are essential steps in preparing data for training machine learning models.

Scikit-learn:

Scikit-learn is a widely used machine learning library in Python that offers a comprehensive suite of tools for various machine learning tasks, including classification, regression, clustering, and dimensionality reduction. In the fraud detection project, scikit-learn is utilized for training and evaluating the fraud detection model. It provides a variety of machine learning algorithms, including logistic regression, which is often used for binary classification problems like fraud detection. Scikit-learn also offers utilities for data preprocessing, feature scaling, model selection, and evaluation. It supports common evaluation metrics such as accuracy, precision, recall, F1-score, and provides functionality for cross-validation to assess model performance. Additionally, scikit-learn integrates well with other libraries, such as Pandas and NumPy, making it a powerful tool for building and evaluating machine learning models in the fraud detection project.

Project Description

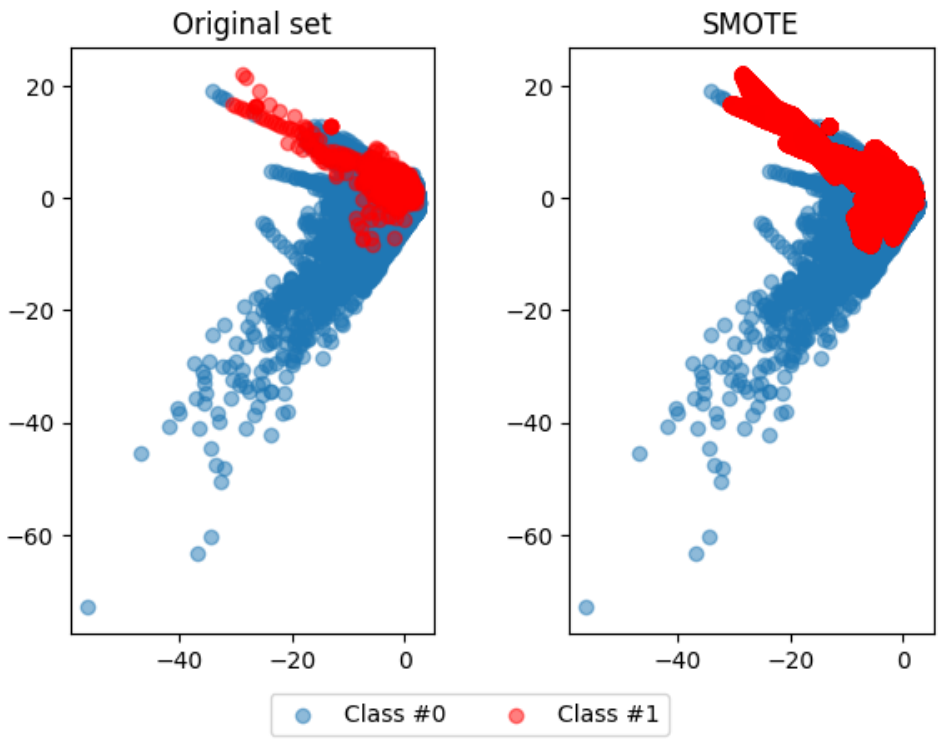
Dataset Exploration and Preprocessing

The dataset contains the following variables:

- Numerically encoded variables V1 to V28 which are the principal components obtained from a PCA transformation. Due to confidentiality issues, no background information about the original features was provided.
- The Amount variable represents the transaction amount.
- The Class variable shows whether the transaction was a fraud (1) or not (0).
-

By their nature, fraud occurrences are fortunately an extreme minority in any list of transactions. However, machine learning algorithms usually work best when the different classes contained in the dataset are equally present. Otherwise, there's little data to learn from. This problem is called the class imbalance.

To fix imbalance, we can re-balance our data using the synthetic minority oversampling technique (SMOTE). Unlike random oversampling, SMOTE is slightly more sophisticated since it does not just create exact copies of observations. Instead, it uses characteristics of nearest neighbours of fraud cases to create new, synthetic samples that are quite like the existing observations in the minority class.



Model Training and Evaluation

n this fraud detection project, a machine learning model was created using logistic regression and SMOTE (Synthetic Minority Over-sampling Technique). Logistic regression is a widely used algorithm for binary classification tasks, making it suitable for detecting fraudulent transactions. The model was trained on a dataset containing historical transaction data, with the target variable indicating whether a transaction was fraudulent or legitimate. To address the issue of imbalanced data, where fraudulent transactions are significantly fewer than legitimate ones, SMOTE was applied. SMOTE generates synthetic samples of the minority class (fraudulent transactions) to balance the dataset, thereby improving the model's ability to accurately detect fraudulent activities. The combination of logistic regression and SMOTE helps create a robust and effective fraud detection model for identifying potential fraudulent transactions.

After the logistic regression model was trained using SMOTE and evaluated, it was saved as a pickle file. Saving the model allows for easy reusability and portability. The pickle file contains the trained model's parameters and configuration, enabling it to be used later in the data processing pipeline for real-time fraud detection. By loading the model from the pickle file, the system can make predictions on new transactions efficiently and accurately. This approach ensures that the trained model remains consistent and can be seamlessly

integrated into the fraud detection pipeline whenever new data needs to be processed and evaluated for potentially fraudulent activities.

Without SMOTE

Flagged Fraud	0.0	1.0
Actual Fraud		
0.0	85285	11
1.0	56	91

With SMOTE

Flagged Fraud	0.0	1.0
Actual Fraud		
0.0	83201	2095
1.0	12	135

Pipeline with Prefect

Prefect is well-suited for handling real-time data in several ways:

Workflow Management:

Prefect provides a powerful workflow management system that allows you to define, schedule, and manage the execution of tasks. This enables you to create complex data processing pipelines that can handle real-time data streams effectively.

Task Dependencies and Parallelism:

With Prefect, you can define dependencies between tasks, ensuring that they execute in the correct order. This is crucial when processing real-time data that may have dependencies on previous or concurrent tasks. Prefect also supports parallel execution, allowing tasks to run concurrently when possible, thus improving the overall performance of the pipeline.

Scheduling and Triggering:

Prefect offers various scheduling options, allowing you to define when and how often a pipeline should be executed. This is particularly useful for real-time data processing, where you may want to process data at specific intervals or in response to certain events or triggers. Prefect supports interval-based scheduling, cron-based scheduling, as well as event-based triggers.

Fault Tolerance and Retries:
Real-time data processing pipelines must be resilient to failures and disruptions. Prefect provides built-in fault tolerance mechanisms, including automatic retries and error handling. If a task fails, Prefect can automatically retry it a specified number of times or until it succeeds. This ensures that the pipeline can handle transient failures and continue processing real-time data seamlessly.

Monitoring and Visibility:
Prefect offers a rich set of monitoring and visualization tools, allowing you to track the progress, status, and performance of your data processing pipeline in real-time. You can monitor task execution, view logs, and receive notifications or alerts when specific conditions are met. This level of monitoring and visibility is crucial for ensuring the reliability and accuracy of real-time data processing.

Integration with External Systems:
Prefect integrates well with a wide range of external systems and libraries commonly used in real-time data processing. This includes data streaming frameworks like Apache Kafka or Apache Flink, databases, cloud storage services, and more. Prefect provides dedicated integrations and utilities that make it easy to incorporate real-time data sources and sinks into your pipeline.

Overall, Prefect's flexible workflow management, task dependencies, scheduling capabilities, fault tolerance mechanisms, monitoring tools, and integrations make it highly suitable for real-time data processing. It enables you to build robust and scalable pipelines that can handle the complexities and challenges of real-time data streams effectively.

Here is an explanation of the code:
Importing libraries:

- Pandas is imported as pd for data manipulation.
- Time delta is imported from datetime to define time intervals for scheduling.
- Necessary modules from Prefect are imported for task and flow creation.
- pickle is imported to load the trained model from a pickle file.
- time is imported to introduce delays between pipeline runs.

Task Definitions:

- The *read_csv_file* task reads data from a CSV file and returns a DataFrame.
- The *load_model* task loads the trained model from a pickle file.

- The *preprocess_data* task preprocesses the input data by selecting relevant columns.
- The *predict_data* task uses the loaded model to predict fraud on the processed data.
- The *write_csv_file* task writes the processed data and predictions to an output CSV file.
- The *combine_data* task combines the processed data, actual labels, and predictions into a single DataFrame.

Flow Definition:

- The *process_data* flow is defined using the @flow decorator.
- File paths for input and output CSV files are defined.
- The *read_csv_file* task is called to read data from the input CSV files.
- The *preprocess_data* task is called to preprocess the data.
- The *load_model* task is called to load the trained model.
- The *predict_data* task is called to predict fraud using the loaded model.
- The *combine_data* task is called to combine the data, actual labels, and predictions.
- The *write_csv_file* task is called to write the combined data to the output CSV file.

Running the Pipeline:

- The pipeline is executed using a loop that runs the process_data flow multiple times.
- A delay of 5 seconds is introduced between each pipeline run using time.sleep().

This pipeline continuously reads data from two CSV files, preprocesses the data, applies the loaded model to predict fraud, combines the predictions with other data, and writes the results to an output CSV file. The pipeline runs every 5 seconds in this example, but the timing can be adjusted based on the project requirements.

Flask Web App

Flask is a lightweight web framework in Python that is well-suited for building data pipelines due to the following reasons:

Integration with Python:

Flask is written in Python and seamlessly integrates with the Python ecosystem. This makes it easy to incorporate other Python libraries, tools, and modules commonly used in data processing and analysis, such as pandas, scikit-learn, or TensorFlow. Flask allows you to leverage the power of Python to handle data processing tasks within your pipeline.

Data Visualization and Monitoring:
Flask's template engine allows you to render dynamic HTML templates, making it easy to create visually appealing and interactive dashboards to visualize pipeline results, metrics, or monitoring information. You can incorporate charting libraries like Plotly or Bokeh to create data visualizations and provide real-time insights into your data pipeline.

Easy Deployment and Scaling:
Flask applications can be deployed easily on various hosting platforms and cloud providers. Python's extensive deployment options, such as containerization with Docker or deployment on serverless platforms like AWS Lambda, make it straightforward to scale and manage your data pipeline as the data volume or processing requirements grow.

Flask Extensions and Libraries:
Flask has a vibrant ecosystem of extensions and libraries that can further enhance your data pipeline capabilities. For example, you can use Flask-SQLAlchemy for database integration, Flask-Cache for caching data, or Flask-RESTful for building more complex REST APIs. These extensions provide additional functionality and simplify the implementation of advanced features in your data pipeline.

Asynchronous Task Execution:
Flask supports asynchronous task execution, which is valuable in data pipelines where certain tasks may require longer processing times or involve external services. By leveraging libraries like Celery or Redis Queue, you can offload time-consuming tasks to background workers, ensuring that your Flask application remains responsive and can handle concurrent requests efficiently.

Testing and Debugging: Flask's lightweight nature and modular design make it easy to test and debug your data pipeline code. The simplicity of Flask allows you to write unit tests for individual components or modules, ensuring the correctness

of your pipeline's logic. Flask's built-in debugger and error handling capabilities also facilitate identifying and fixing issues during development and deployment.

Here's an explanation of the code:
Importing Libraries: The necessary libraries for Flask, CSV handling, and time are imported.

Flask App Initialization: An instance of the Flask application is created with the Flask(__name__, static_url_path='/static') line.

CSV File Path: The path to the output CSV file generated by the fraud detection pipeline is defined.

classify_last_n_lines Function: This function takes a file path and the number of lines to classify as inputs. It reads the last n lines from the file, classifies each line based on the values in the last two columns, and returns a list of classifications.

get_data Route: This route is triggered when the /get_data endpoint is accessed. It calls the classify_last_n_lines function with the CSV file path and the desired number of lines to classify. The result is returned as the response.

search_csv_file Function: This function takes a filename and a search value as inputs. It reads the CSV file line by line and searches for a row where the first column matches the search value. If a match is found, the row is returned; otherwise, None is returned.

transaction_details Route: This route is triggered when the /transaction_details endpoint is accessed with a POST request. It retrieves the transaction ID from the form data, calls the search_csv_file function to find the details of the transaction in the CSV file, and renders the transaction.html template with the transaction details.

display_csv Route: This route is the home page route ("/"). It calls the classify_last_n_lines function to get the latest classifications from the CSV file and renders the home.html template with the result.

Flask App Execution: The Flask application is run with the app.run(debug=True) line, allowing the web app to be accessed and debug information to be displayed.

In summary, this Flask web application provides functionality to display the latest fraud classification results, search for transaction details, and visualize the data processed by the fraud detection pipeline.

Conclusion

In conclusion, the fraud detection project combines Prefect, pandas, scikit-learn, and Flask to create a robust and efficient system. The project demonstrates the ability to generate, process, and analyze real-time data for fraud detection. By leveraging Prefect's workflow management capabilities, pandas' data manipulation functions, scikit-learn's logistic regression model, and Flask's web application framework, the project successfully detects fraudulent transactions and provides a user-friendly interface for monitoring and investigating suspicious activities. The project serves as a foundation for further enhancements and showcases the potential of leveraging technology to build intelligent systems that ensure the security of financial transactions.