IntelliHack 5.0

Machine Learning hackathon

Q4 - Report 02

The Code Rushers

1 System Architecture Design

Stock markets are one of the most important financial pillars in the country's economy. They play a crucial role in a country's success. There's a huge risk in inventing these, and there is also high security involved. Therefore, the existence of a decision support system for evaluating markets' conditions and providing suggestions to investors can be very beneficial.
Below is a proposed system architecture for a Stock Price Prediction system.

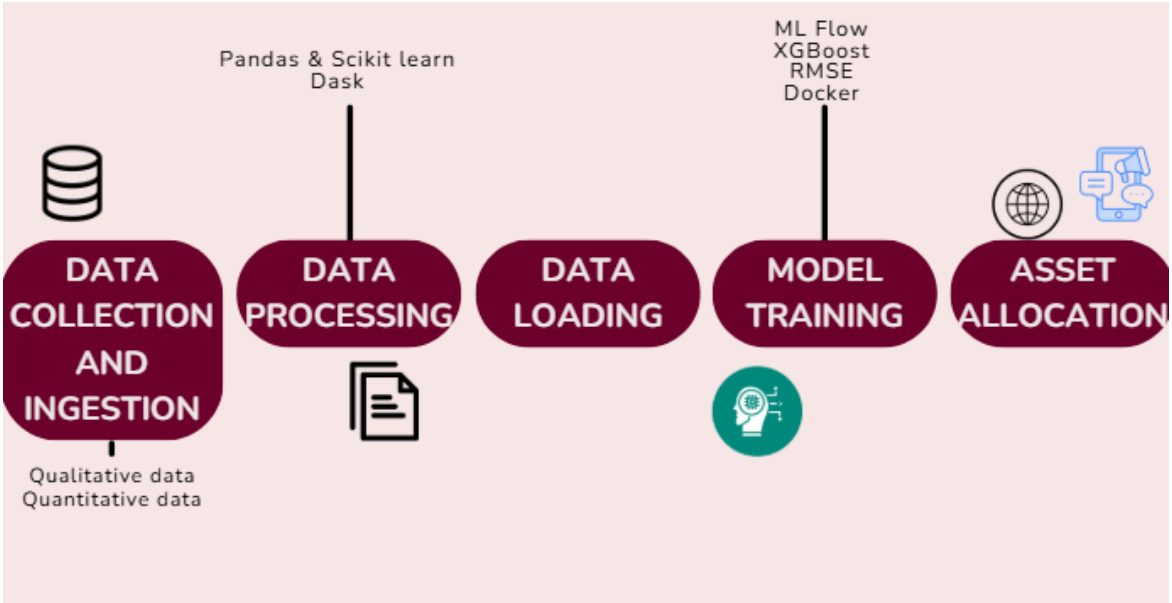

Figure 1: System Architecture

1.1 Data Collection & Ingestion

The system is based on qualitative and quantitative data for deciding between buy, keep, or sell options. The quantitative data includes past stock trades data of the company, and the qualitative data is gathered from expert traders of the stock market.
Example: Alpha Vantage API, Yahoo Finance API, Quandl API
This works as Batch Data ingestion. Data is queried daily and put into a SQL database. Data can be fetched and stored at regular intervals for subsequent processing.
Technologies: **Airflow** for scheduling tasks, **Google Cloud Functions**.

1.2 Data Processing

Cleaning: Handling missing values using Forward fill and interpolation, outliers are removed.
Feature Engineering: Stock price prediction is done using OHLC price data along with the volume of trades.

Scaling: Standardize or normalize numerical features.
Tools and Technologies:

- Pandas and Scikit-learn: for smaller data and feature engineering.
- Dask: For scaling Pandas-like workflows for larger datasets.

Data Storage Architecture

Raw Data Storage: SQL database for storing raw stock data.
Processed Data Storage: Stored data is then converted to JSON files.

1.3 Data Loading

The third step is for loading data from stored JSONs and merging sequential reports published quarterly. This stage speeds up the data processing and acts as a caching system.

1.4 Model Training

Model Training & Evaluation:

- Automated pipeline to retrain the model after every ingestion.
- MLflow is used for experiment tracking and versioning of models.
- XGBoost is trained on the processed data.

Model Deployment: Deploy the model in a Docker container to isolate the environment and ensure portability. Google AI Platform can be used for model deployment, monitoring, and scalability.

Model Monitoring:

- Implement continuous monitoring using Prometheus to track metrics like prediction accuracy, model performance over time, and latency.
- Use Grafana to visualize model performance and detect any potential data drift or performance degradation.
- Data Drift Detection: Use statistical tests to monitor whether the distribution of incoming data changes over time, indicating the need to retrain the model.

Model Evaluation & Retraining:

- Regularly evaluate the model on a holdout validation dataset.
- Track performance metrics like RMSE, Directional Accuracy, and Simulated Trading Performance.
- Automate retraining if performance metrics fall below a predefined threshold, triggered by the monitoring system.

1.5 Asset Allocation (Insight Delivery)

A website and an SMS system can be incorporated to give the traders real-time updates.

API Gateway Expose the model's predictions and insights via a REST API using Flask. The API handles requests from analysts, brokers, or trading systems and returns predictions or trading recommendations (e.g., "buy" or "sell").
Real-Time Dashboard A Streamlit dashboard can be used to present stock price predictions, insights, and live data updates. Key metrics such as predicted stock price, volatility, and trading signals are displayed. Provided historical performance comparisons.
Notification System An alert system to notify analysts or brokers when significant changes are predicted (e.g., price spike, trend reversal). Use Twilio API to send SMS notifications, or integrate email systems for alerts.

1.6 System Considerations

Scalability:

- **Data Scalability:** Data filtering and feature selection improves scalability. Batch-wise ingestion and processing makes it more scalable. Usage of distributed storage systems like Apache Hadoop or cloud storage (AWS S3, Google BigQuery).
- **Computational Scalability:** Optimizing code using parallel processing (NumPy, TensorFlow, PyTorch). Deploy serverless computing (AWS Lambda, Google Cloud Functions) for real-time predictions.
- **Model Scalability:** Usage of XGBRegressor for long-term sequential data.

Reliability:

- **Data Reliability:** Usage of multiple data sources to cross-check information. Data cleaning techniques (handling missing values, outlier detection). Real-time anomaly detection to filter unreliable inputs.
- **Risk Management & Reliability:** Continuously retrain models with updated market data. Integrate stock market movement prediction.

Latency Requirements: Parallel Processing & Asynchronous Execution (e.g., Dask, Ray for Python). Optimizing real-time trading models.

Cost Considerations:

- Usage of free or open-source datasets (Yahoo Finance, Quandl - limited options).
- Usage of serverless computing (AWS Lambda, Google Cloud Functions) for on-demand execution.

- Training models in batches instead of real-time retraining.
- Implementing batch inference for non-real-time scenarios.

2 Component Justification

2.1 Data Collection & Ingestion

Technology/Approach:

- API-based Integration (e.g., Alpha Vantage, Yahoo Finance)
- Historical data from the company.
- Batch-wise Ingestion

Why this solution?

- APIs provide an easy-to-integrate, reliable, and structured way to fetch financial data. These services offer historical, real-time, and sometimes alternative data with well-documented endpoints.
- Historical data provide trends related to the particular company.

Trade-offs:

- API-based solutions often come with usage limitations (e.g., rate limits) and costs (especially for real-time data). You may need to pay for premium plans for continuous data.

2.2 Data Processing Pipeline

Technology/Approach:

- Pandas and Scikit-learn
- Dask for in-memory data processing for smaller datasets.

Why this solution?

- Pandas/Dask is well-suited for smaller datasets and provides intuitive APIs for data manipulation and transformation.

Trade-offs:

- Dask is a great option for parallel processing on smaller machines but might struggle with very large datasets.

2.3 Feature Engineering

Technology/Approach:

- Time-Series Feature: Moving averages, Volatility, Exponential Moving Averages
- Lag features

Why this solution?

- Time-series indicators such as moving averages are widely accepted in financial analysis and provide actionable insights.

Trade-offs:

- Time-series feature engineering can be computationally expensive, especially if you are incorporating multiple technical indicators or features.

2.4 Model Training & Evaluation

Technology/Approach:

- LSTM (Long Short-Term Memory) for sequence modeling.
- Random Forest/Gradient Boosting for non-linear patterns.
- XGBoost for boosting performance and feature importance analysis.

Why this solution?

- LSTMs are a popular choice for time-series forecasting as they can capture long-term dependencies in data.
- XGBoost/Random Forest are powerful ensemble models that can handle tabular data well and are particularly useful for features generated through traditional technical indicators.

Trade-offs:

- LSTM requires more computational resources for training and is more time-consuming than simpler models.
- Ensemble models like XGBoost may perform well in terms of accuracy, but they are prone to overfitting if not tuned properly.

2.5 Model Deployment & Monitoring

Technology/Approach:

- Docker for containerization of models and ensuring reproducibility.
- Model Monitoring tools like MLflow for tracking model performance, drift, and retraining.

Why this solution?

- Docker makes it easier to deploy the model in different environments (e.g., cloud, on-premises) without worrying about dependencies.
- MLflow can help monitor model drift and performance in real-time, ensuring the model stays relevant and continues to perform as expected.

Trade-offs:

- Docker adds complexity to the deployment process, especially in smaller teams or organizations.
- MLflow requires proper configuration and integration with your model pipeline, which can be time-consuming. Over-monitoring can also add unnecessary overhead.

2.6 Insight Delivery to End-Users

Technology/Approach:

- Web-based Dashboards using Dash, Streamlit for visualizations and prediction insights.
- SMS notification system.
- APIs for delivering model predictions to analysts/brokers.

Why this solution?

- Dash/Streamlit allows for the creation of interactive web dashboards with Python, making it easy to visualize stock price trends and predictions in real-time.
- APIs provide a clean, accessible interface for programmatic access to predictions and insights.

Trade-offs:

- Web dashboards may require continuous development and maintenance as new features are added or user requirements change.
- API delivery might require additional security measures, such as rate limiting and authentication to avoid abuse.

3 Data flow

3.1 Data Flow explanation

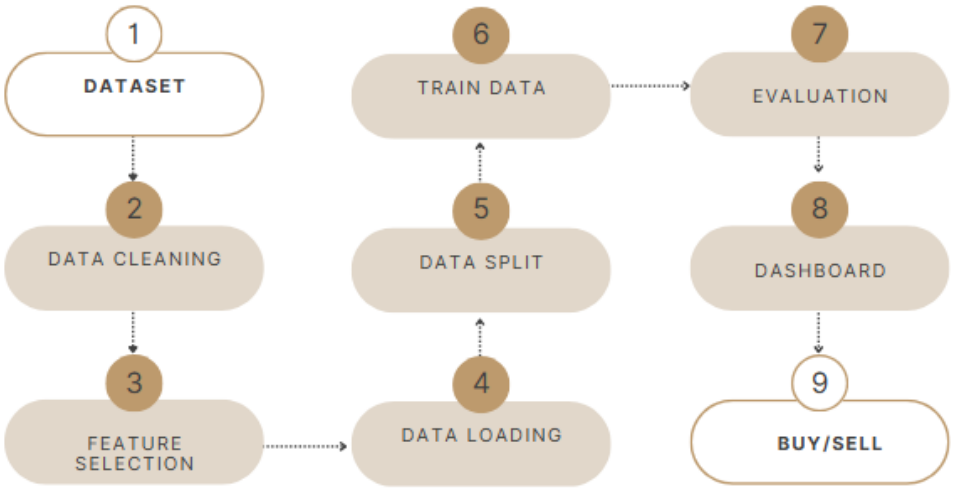

Figure 2: Data flow diagram

1.Data Gathering: The data gathering step includes crawling and saving reports' announcements (online and historical data) in a SQL database. After downloading announcements, the next step is saving the financial reports' data in their original format (HTML files).

2.Data Cleaning: In this step, the main activity is cleaning of the semi-structured reports, including normalizing texts and numbers, mapping reports to a unified format, and etc. For later access to the cleaned data, we save it in JSON format in the SQL database.

3.Data Loading: The third step is for loading data from stored JSONs and merging sequential reports published daily. This stage speeds up the data processing and acts as a caching system.

4.Stock Modeling: The main part of the system is training a model that can predict stock return in various periods. For this purpose, different categories of features are prepared for predicting stock return and a machine learning model is trained.

5.Asset Allocation: The last step is for allocating assets in different securities. The trained model in the previous step is used for predicting the stock return and then determining the percentage of fund for each asset.

The main point of system design is the pipeline architecture, which allows the output of each stage to be stored independently, and each stage can be run separately.

3.2 Stock Market Movement Prediction

The stock market's health is a key indicator of overall economic well-being, and the good performance of stocks attracts investors' attention to the stock market among different financial markets. Asset allocation, in general, is the decision of distributing funds in different assets of various financial markets. Asset allocation in various markets is a key decision for every investor. With asset diversification, the investment risk is minimized in terms of the earned return.
The system's core functionality is based on the proposed stock market return predictor.
Based on the computed weighted probabilities, we choose between two asset allocation strategies:

1. Market growth probability of more than 50%
2. Market growth probability of less than 50%

4 Challenge Analysis

1.Gathering, cleaning, integrating, and exploring published fundamental data. Solution:

- Use data cleaning techniques (e.g., handling missing values, outliers).
- Explore alternative data sources like sentiment analysis or economic indicators.
- Continuously monitor data quality and clean it during data preprocessing.

2.Some companies in the stock market publish their financial reports regularly in each quarter. But there are more challenges in accessing those data. A crawler is needed for continuous download of announcements. Data is published in semi-structured data, populated with texts and numbers. Data format is changed in different quarters, thus requiring a data mapper for unifying formats. Stocks in different industries have different report formats.

3.Latency & Real-Time Prediction - High latency can result in missed opportunities or incorrect predictions, especially in the highly volatile and fast-paced stock market. **Solution:**

- Optimize the data ingestion pipeline for faster real-time data processing.
- Use model quantization or simpler models for quicker inference.

4.Market Volatility and External Factors - Stock prices are affected by historical data and external factors such as geopolitical events, natural disasters, or economic crises. Predictive models based only on historical prices may fail to adapt to these sudden changes.
Solution:

- Integrate external data sources (e.g., news sentiment, macroeconomic indicators) into your model to improve prediction robustness.
- Regularly retrain the model with fresh data to capture recent market changes.

5.Computational Resource Requirements - Training sophisticated models like LSTMs or transformers requires significant computational power (GPUs/TPUs), which can lead to high infrastructure costs.
Solution: Usage of simple models like XGBRegressor, which need less computational power.

References

1. https://arxiv.org/html/2412.05297v1
2. https://www.researchgate.net/publication/356093237_Stock_Market_Prediction_Using_Machine_Learning_Techniques