IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Digital Object Identifier

Applying Predictive Analytics to Financial Markets

COREA KITTII[1], ARON SAMAYOA[2], AND DANIEL CASCO.[3]

ABSTRACT Within this paper we present a price prediction application that utilizes tick-level price data from financial markets in order to perform predictive analytics by way of machine learning models, and that is currently under development. The application includes a data importation tool written Python that makes use of both the PyArrow and Pandas libraries in order to import data from both CSV files and Parquet files, which is then both stored in and managed by a MySQL database. Two types of machine learning models are planned to be implemented; a Long-Short-Term Memory model and a Transformer model. These models can be trained on specific securities in order to perform predictive analysis. Instances of fully trained models can be saved for each security in the database, where they can be retrieved for future use. At present a basic LSTM model has been implemented that is capable of training on the chosen dataset. Further progress has been limited due to time constraints and the computational cost of training the model. It currently takes between hours and days to train the current implementation of LSTM model, depending on the volume of data, which makes fine-tuning and other adjustments difficult to perform with the allotted timeframe. Regarding the UI, this application provides two interfaces: a user interface and an administrator interface. Users are capable of viewing candlestick charts that are formed out of the tick-level price data, and can view the results of the predictive analysis performed by the machine learning models. Administrators can do the same, while also having the ability to initiate training sessions for the models, as well as any required maintainance of the database. Once completed, this application can be used to predict future price movements of financial markets simply, quickly, and accurately.

INDEX TERMS Algorithmic Trading, Financial Markets, LSTM, Machine Learning, MySQL Database, Pandas, Price Prediction, PyArrow, Python, Tensorflow, Tick-level Data, Transformer Models, React Native

I. INTRODUCTION

IT is generally understood that accurately predicting the future is extraordinarily difficult, if not impossible; however, that hasn't stopped people from trying, especially in regards to financial markets. Of all the research that has been performed on these markets, predicting future price movements has proven to be one of the most challenging problems due to the variables involved, such as market volatility, investor sentiment, and factors external to the financial market, which include the economy, climate, politics, and more. Where simpler models often fall short due to the complexities of financial time series data, machine learning techniques, particularly deep learning, have shown promise. Of the many different types of deep learning models, Gated Recurrent Units (GRU), Long-Short-Term Memory (LSTM), and Transformer models have displayed promising performance, with our application utilizing the latter two. Extensive research has been performed on this subject, which has explored a wide variety of machine learning techniques in order to find those suitable for price prediction. These range from the more classic support vector machines (SVM) all the way to the more advanced neural networks. Studies have proven that using deep learning models for price prediction is effective, while determining which models are the most effective is an ongoing process [1]. Precision price prediction has had it's importance proven via advancements made in algorithmic trading, especially in high-frequency trading, where decisions must be made based on real-time analysis of the market [2]. Further refining the accuracy of price predictions can be done by utilizing tick-level price data, which can capture even the smallest market movements due to it's granularity [3].

Past research has performed comparisons on the performance of GRU, LSTM, and Transformer models in regards to price prediction, with the dataset consisting of Tesla (TLSA) price data ranging from 2015 to 2024. The dataset underwent preprocessing in order to correct for missing values and to normalize price fluctuations, along with other adjustments

that allowed for the optimization of training each model. The results of the experiments that were performed showed that LSTM models performed better than both GRU and Transformer models, while Transformers performed the worst out of the three [1].

II. PROJECT OVERVIEW

Our application will utilize historical tick-level price data from the financial market in order to perform predictive analytics using machine learning models, with the application also providing a tool to import said data. Users will be provided an interface that allows them to view the price data in the form of candlestick charts, as well as view the results of the predictive analysis produced by the machine learning models. Administrators are capable of doing that on top of having the ability to both initiate model training and perform necessary maintainance.

A. SYSTEM DESIGN

The system is designed in a modular fashion, and features the data importation tool, the database, the two types of machine learning models, an interface for administrators, and an interface for normal users. A more detailed description is as follows:

1) Data Importation Tool

The data importation tool is implemented using the Python programming language, which allows us to take advantage of it's excellent third-party library support. We utilize both the PyArrow and Pandas libraries, as well as various built-in libraries. The main purpose of this tool is to convert financial tick-data stored in compressed CSV files, process it, and store it within the MySQL database. This tool also generates and stores OHLCV format candlestick data and stores is within the database as well.

The importation tool works as follows:

1) File Discovery: A user-specified directory is searched for ZIP archives which are assumed to contain CSV files.
2) Processing ZIP Files: For each ZIP file found within the user-specified directory, it's content's are extracted to memory.
3) CSV Conversion: Using PyArrow, the data within each CSV file is first read into a PyArrow Table where unnecessary columns are removed and the existing ones are renamed.
4) Timestamp Normalization: All timestamps are converted to nanosecond precision, as their precision otherwise varies depending on the age of the data. For the candlestick data the timestamps are integers, while timestamps for the tick-data are converted to fixed point numbers to make normalizing the timestamp during the preprocessing step more accurate.
5) OHLCV Generation: OHLCV formatted data is automatically generated from the tick-data for multiple timeframes (e.g. 1min, 2min, 3min, ..., 4h, 1D, 7D). Pandas' "resample()" function is used to aggregate this data efficiently.
6) Database Integration: The tool connects to the MySQL database via the "mysql.connector" library. Necessary tables are created dynamically, and data is inserted in batches of 500,000 rows, which avoids sending packets larger than what a MySQL database supports by default. Both PyArrow and Pandas are used to facilitate the processing of batches and insertion of data into the database.
7) Parquet Support: Importing data from Parquet files is planned, but currently remains unimplemented.

This importation process is an important first step in populating the database with the financial tick-data necessary for training machine learning models, as well as the candlestick data necessary for generating charts for human viewing and analysis.

2) MySQL Database

Once imported, data is stored within a MySQL database for easier access. Candlestick charts are generated using this data, and it is also fed into machine learning models during their training. Each instance of a fully trained model is also stored within the database, allowing for easy retrieval. The database also stores the results of predictive analysis so that they can be viewed by users.

Data within the database is stored according to the following schema:

a: Tick-Level Data

Price data for each security is stored in a table whose name is in the format TICK_DATA-[security].

- trade_id: A unique ID is assigned to each trade
- time: Unix-timestamp of when the trade took place
- price: The value of the security at the time of the trade
- volume: How much of the security was traded
- side: Whether the trade was a buy or sell order

b: Candlestick Charts

Candlestick charts for each security is stored in a table whose name is in the format CANDLE_[timeframe]-[security].

- candle_id: A unique ID is assigned to each trade
- open_time: Unix-timestamp of the first trade during the time frame
- close_time: Unix-timestamp of the last trade during the time frame
- open_price: Value of the security at the start of the time frame
- high_price: Highest value the security reached during the time frame
- low_price: Lowest value the security reached during the time frame
- close_price: Value of the security at the end of the time frame
- volume: How much of the security was traded during the time frame

c: Model Instances

All trained models instances are saved in the database in the MODEL_INSTANCES table.

- security: The security the model was trained on
- model_id: A unique ID assigned to each model saved in the database
- model_type: Whether the model is an LSTM or a Transformer
- date_trained: Timestamp of when the model was trained

d: User Information

User information is saved in the USER_INFO table.

- user_id: A unique ID is assigned to each user; no duplicate ID's are allowed, and this field cannot be null.
- first_name: The user's first name; this field cannot be null, but duplicate values are acceptable.
- last_name: The user's last name; this field cannot be null, but duplicate values are acceptable.
- username: Each username must be unique; this field cannot be null.
- password: Contains the password hash. Each password must be at least a minimum length, and feature a mix of alphanumeric and special characters. This field cannot be null.
- admin_status: Set to true or false depending on whether the user has been granted administrator rights.

3) Machine Learning Models

Within our application we plan to implement two types of machine learning models:

- Long-Short-Term Memory (LSTM) models, which excel at performing predictive analysis on time-series data that spans a long period of time.
- Transformer models, which perform better at predictive analysis on time-series data that spans a shorter period of time, and are generally faster to train than LSTM's.

Models are trained on specific securities to improve the accuracy of their predictions; in other words, each instance of a fully trained model is a specialist in the security it was trained on. We have currently implemented a basic LSTM model that is capable of both training and testing on our dataset; at present, however, we are running up against computational limits when training the model. Training times currently range from approximately 4 hours to multiple days depending on the volume of the currency pair the model is being trained on. Experimenting with different hyperparameters in order to fine-tune the model is currently prohibitively expensive from a time perspective. Because of this, the Transformer model currently remains unimplemented.

Regarding data preparation for the machine learning models, we've implemented a preprocessing pipeline. This pipeline normalizes the 4 features contained within the data that the model trains on (time, price, volume, side) to a value between 0 and 1, as leaving the data as is would cause the comparatively extreme values like the timestamp to drown out everything else, making training the model properly impossible.

The preprocessing pipeline is as follows:

1) Database Connection and Identifying Data Range: The script first connects to the MySQL database that houses the financial tick-data contained within the corresponding "TICK_DATA" table (e.g. "ETHUSDC_TICK_DATA"), as well as the minimum and maximum "trade_id" values, which are used in a later step.
2) Train/Test Splitting: The tick-data for the chosen currency pair is split into a training set and a testing set, with the former comprising 60% of the data, and the latter comprising the remaining 40%. The indices that are used define this split are saved for the sake of reproducibility.
3) Data Fetching: In order to avoid excessive memory usage, data is fetched from the database in chunks, with the trade ID being used to access each chunk directly, rather than utilizing MySQL's inefficient "OFFSET" keyword.
4) Data Normalization: The features that make up the dataset (time, price, volume and side) are normalized in order to facilitate the training process. "MinMaxScaler" from the third-party library "scikit-learn" is utilized for this purpose.
5) Parallel Preprocessing: Data fetching and preprocessing makes use of multiple worker threads in order to speed up the process. Each worker fetches and processes data from a predetermined range of trade ID's, and each worker writes the results to their own file, with Parquet files being used for space efficiency.
6) Final Aggregation and Cleanup: Once all chunks have been processed, the intermediate Parquet files created by each worker are merged together. Afterwards, each intermediate file is then deleted, as they are no longer necessary.

Though there is still room for improvement, this preprocessing pipeline represents an important step in the process of training machine learning models that can accurately perform price prediction. More advanced preprocessing techniques could be implemented in the future, which would further increase the quality of the data and the effectiveness of the models and the training process.

As mentioned before, the only model we currently have implemented, the LSTM model, is at a level of basic functionality, and it's usefulness is currently limited. At present, the model consists of two LSTM layers, along with dropout regularization, and is designed to capture any time-based dependencies in the preprocessed financial tick-data.

The LSTM model works as follows:

1) Data Loading: Parquet files containing the preprocessed training and testing data are loaded into memory.
2) Sequence Creation: Sequences of data points, which in this case means 60 ticks or trades, serve as input for

the model, with it's prediction of the next tick being the output. A number of batches of these sequences are generated and then saved to disk in compressed NumPy files for both the training and testing data. This is done to avoid excessive memory usage.

3) Memory Management: Due to the size of the datasets batches are also utilized during model training and testing. Batches are loaded from disk into a queue, and no more than eight batches are allowed to be loaded at any given time.
4) Model Definition: The LSTM model is a sequential model, and consists of two LSTM layers, two dropout layers, and a dense layer which outputs the predicted price.
5) Training Process: The LSTM model trains on each batch that's loaded from disk one after the other, and trains on the data for one epoch (i.e. one single pass through the entirety of the training data). In the future, training for more than one epoch would be more effective, as would loading more data into memory at once, given more time and computational resources.
6) Model Evaluation: The models performance is evaluated post-training, with the testing data being batched in a similar process.
7) Inverse Normalizing and Metrics: The normalization process needs to be reversed before outputting the predicted prices and calculating metrics that measure the model's peformance. The metrics currently used are Mean Squared Error (MSE) and Mean Absolute Error (MAE). For these metrics, the lower the value the better.
8) Result Storage: The results of the model's predictions on the testing data are stored in the MySQL database, and the trained model is saved to disk.

As it currently stands, this model is still in an early stage of development, with plenty of room for improvement. In the future it is likely that the model will need additional layers, as it's current configuration may not be enough to completely the complex patterns that make up financial tick-data. A more complex model would, however, increase training times, which is already around four hours for one of the lower volume currency pairs, and likely to be days for the higher volume currency pairs. Training for multiple epochs would be more effective, but would serve as a multiplier for the training times. As such, while this model can produce price predictions, the accuracy is far from optimal.

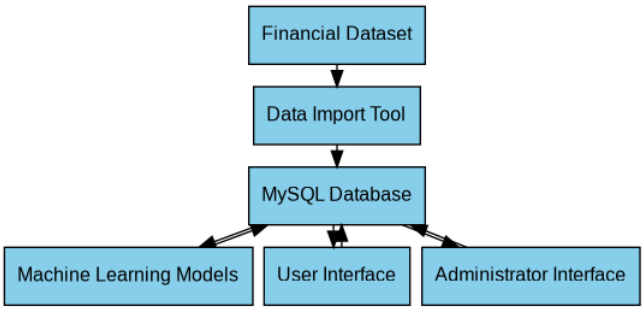
4) User Interface

We use React Native for the user interface, which is a Javascript framework that allows for the development of cross-platform mobile applications, as well as React Native for Web, which allows for the porting of React Native applications to web browsers and greatly expands the number of platforms users can use to access our application. The third-party Python library Flask used as the backend of the user interface, and is used as an intermediary between it and the MySQL database.

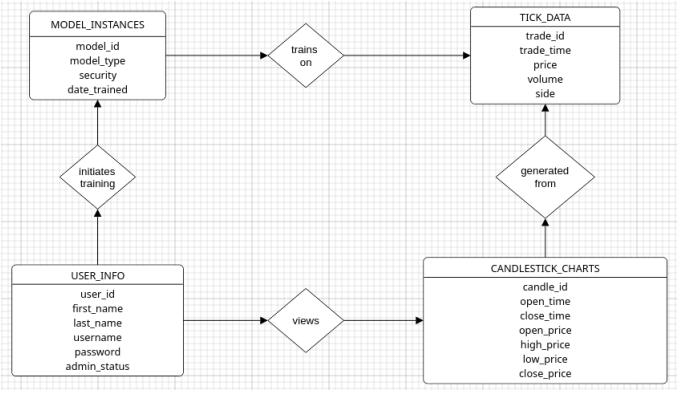
In regards to the UI itself, it has the following functionalities, with differences existing between normal users and administrators:

- View candlestick charts generated from imported tick-level price data.
- View the results of predictive analysis performed by trained machine learning models.
- Initiate model training sessions (Administrator Only).
- Data importation (Administrator Only).
- Database management and maintainance (Administrator Only).
- User account creation, modification, and deletion (Administrator Only).

5) Design Diagram



6) ER Diagram



B. DATASET

Our choice of datasets to use for testing was limited by two main factors: time and cost. While tick-level price data can be obtained from stock exchanges, it is often at a significant cost. The stock exchange IEX does offer historical tick data for free, but it requires spending a significant amount of time parsing out unneeded data, as the files they provide contain everything that happens on their network. We ultimately decided to source our data from the crypto exchange Binance, who provides their tick data in compressed CSV files that can be readily imported into our database. From their historical data repository at data.binance.vision, we selected five cur-

rency pairs with volume levels ranging from high to moderate from their spot trading data:

- BTC/USDT (Bitcoin & Tether)
- ETH/USDT (Ethereum & Tether)
- ETH/BTC (Ethereum & Bitcoin)
- BTC/USDC (Bitcoin & USD Coin)
- ETH/USDC (Ethereum & USD Coin)

C. DISCUSSION

In this section, we shall discuss in depth the potential advantages of our application, as well it's limitations. We'll also discuss how it fares against similar applications and services.

1) Advantages and Limitations

The primary advantage our application has when compared to other price prediction solutions is the use of tick-level price data. Training our machine learning models on data with a greater level of granularity allows for them to gain greater insight on how the financial market functions, as opposed to using OHLC price data. The creation of an OHLC candlestick involves throwing out most of the information that was generated during the time period the candlestick represents, with that discarded data potentially representing hundreds or thousands of individual trades. That does mean, however, that the storage requirements for tick data are drastically higher than OHLC data. Additionally, acquiring tick data can be prohibitively expensive, especially when it concerns the stock market (IEX is an exception, though it has it's own issues, as mentioned earlier). For business with deep pockets or existing subscriptions to providers of tick data, however, this may be a non-issue.

As it currently stands, our application does have some limitations that should be kept in mind. Our use of machine learning gives us an edge when it comes to predictive analytics, but it does require significant computing resources in order to both train and run them. Additionally, training these models takes time, and insufficient computing resources can extend the time required. Case in point, at present we have only been able to implement a basic LSTM model, with the Transformer model remaining unimplemented, due to the computing resource and time requirements. This also means that the accuracy of this basic model's predictions remains quite low, and serves primarily as a proof-of-concept at this current stage of development. We have also been unable to personally verify the performance of an LSTM model versus a Transformer model in matters of price prediction. As mentioned earlier, our use of tick-level price data means that storage requirements are significantly higher compared to OHLC data, even when utilizing compression. As an example, in our dataset one CSV file decompresses from 1.2 MiB to 5.5 MiB, increasing 4.5x in size. The entirety of the BTC/USDT portion of our dataset takes up approximately 60 GiB; assuming a 4.5x increase after decompression, it would then take up 270 GiB, and our entire dataset would take up 450 GiB. MySQL's default storage engine, InnoDB, does feature compression features, though we have yet to implement them.

2) Comparison to Competitors

Our application is, or at least is largely, unique, though precise comparison with other price prediction solutions is difficult, as details on their inner workings are sparse and obscured by marketing. To date we have not found an application that allows for the level of control over the process of performing predictive analysis, from data importation to model training to data visualization. Users are not using some machine learning model that was pre-trained by another company, are instead involved in the process of training and fine-tuning a model that can suit their specific requirements. It should be noted that this comparison is based upon our application in it's intended final state, and it's present functionality remains limited.

D. CONCLUSION

The development of this price prediction application has and continues to be a challenging endeavor, but it nevertheless has potential. Machine learning techniques are proven to be effective in performing predictive analysis, and the use of tick-level data will allow our models to gain a more granular insight into price movements than is otherwise possible with OHLC data. The availability of two different model types would also allow users to find the one that works best in their specific situation. That being said, their are potentially steep resource requirements, both in compute and storage, with the former being a major challenge for us currently.

In the future, we will continue to refine the LSTM model, as well as implement the Transformer model, and seek ways to optimize the training process while also improving the quality, effectiveness, and accuracy of fully-trained models. Additional features could be added, such as the import and analysis of real-time data and adding support for sentiment analysis, potentially via the implementation of Large Language Models such as Google's Gemini, Anthropic's Claude, or smaller, more efficient models that can be run locally, such as Alibaba's Qwen, Microsoft's Phi, and others. There is room for optimizations as well; as mentioned before, the model training process could be made more efficient, and MySQL could be replaced with a more flexible database system, such as PostgreSQL. All these additions and improvements would significantly improve our application's capabilities while reducing resource requirements, all the while making it a more compelling option for business's in need of it's feature-set.

In conclusion, this paper serves primarily as a progress report on our ongoing efforts to apply machine learning to financial markets, how it can be effectively used to perform predictive analysis on financial data, and the importance of a larger and more granular dataset, despite the drawbacks. Upon completion, this application can be used in a practical fashion in order to extract value from financial markets, and

it can be used academically in order to further the study and development of machine learning in the realm of finance.

REFERENCES

[1] J. Xiao, S. Bi, and T. Deng, "Comparative analysis of lstm, gru, and transformer models for stock price prediction," arXiv, vol. 2411.05790v1, 2024. [Online]. Available: https://arxiv.org/abs/2411.05790
[2] J. Chen, "Algorithmic trading: Definition, how it works, pros & cons," 2023, accessed: 10 Feb. 2025. [Online]. Available: https://www.investopedia.com/terms/a/algorithmictrading.asp
[3] R. Bhattacharyya, "Tick data - what is it, examples, vs bar & minute data," 2024, accessed: 08 Mar. 2025. [Online]. Available: https://www.wallstreetmojo.com/tick-data/