Project Report on TWS Trading Simulation by Chen Chen, Ruoxi Jin

CHEN CHEN and RUOXI JIN

This report details the development and implementation of a trading simulation system integrating the Interactive Brokers Trader Workstation (TWS). It covers data collection, feature extraction and preprocessing, model development, strategy simulation, and the integration with TWS for simulated trading, providing a comprehensive overview of the trading strategy's effectiveness based on sentiment analysis derived from financial news data.

Contents

Manuscript submitted for review to the 25th ACM Conference on Economics & Computation (EC'24).

1 INTRODUCTION

This project report outlines the design, development, and evaluation of a trading simulation system that integrates with the Interactive Brokers Trader Workstation (TWS). The project aims to harness the power of sentiment analysis, extracting sentiments from financial news to inform trading decisions. This integration allows the simulation of trading strategies in a controlled environment, enabling the analysis of potential trading outcomes without financial risk.

2 DATA COLLECTION

The market-aux.py script is responsible for collecting financial news data from the MarketAux platform. This platform aggregates news from various sources, providing a rich dataset that includes up-to-date financial news headlines.

2.1 Historical Market Data Collection

- Identify Relevant Data: Determine which financial instruments (e.g., stocks, futures, forex) are relevant to your strategy. For the E-mini S&P 500 futures contract example provided, you'll collect price and volume data.
- Data Granularity: Decide on the time frame for the data (e.g., 1-minute, 1-hour, daily). Higher-frequency data allows for more granular analysis but requires more storage and processing power.
- Historical Depth: Determine how far back you need historical data. Longer historical periods can provide more robust training but may not always reflect current market conditions.
- Data Sources: Use the TWS API for market data.

2.2 New Headlines Collection

- Relevance: Focus on news that directly impacts your financial instruments. For the S&P 500, this might include economic indicators, earnings reports, and major geopolitical events.
- Sentiment Analysis: Consider performing sentiment analysis on news headlines to quantify the market sentiment. This can be a feature for your model.
- API Integration: Use APIs provided by news sources or financial data providers. Make sure to handle rate limits and request quotas.

2.3 Methodology

- API Utilization: The script makes use of the MarketAux API to fetch news articles. The API requests are crafted to include parameters such as symbols (e.g., $\hat{G}$SPC for the S&P 500), language preferences, and entity filtering to ensure that only relevant news articles are retrieved.
- Data Scraping and Parsing: Using Python's requests and BeautifulSoup libraries, the script scrapes news headlines, publication dates, and associated metadata. This information is crucial for understanding the context and impact of each news item on market movements.

2.4 Design Choices and Assumptions

- Scraping Frequency: It is assumed that the data needs to be updated regularly; therefore, the script is designed to be run as often as necessary without overloading the API (compliance with API rate limits).
- Data Relevance: The script filters for specific symbols to ensure data relevance to the trading strategies that will be employed.

3 FEATURE EXTRACTION & PREPROCESSING

The script sentiment-analysis-data.py handles the preprocessing and feature extraction needed to prepare the scraped data for model training.

3.1 Text Preprocessing

- HTML Tag Removal: Financial news data scraped from the web often contains HTML markup. Removing HTML tags is crucial for obtaining clean text that contains only the content of interest. This is achieved using regular expressions that identify and strip out HTML tags.
- Lowercasing: Converting all text to lowercase ensures uniformity across the dataset. This is important because many text processing tools treat words differently based on case (e.g., "Bank" vs. "bank"). Lowercasing helps in reducing the feature space and improves the robustness of the model.
- Tokenization: This process splits text into individual words or phrases. It is a fundamental step in text analysis, as it turns a string of characters into tokens which can be individually analyzed. Word tokenization is performed using libraries like NLTK, which can handle complex word separation rules.
- Stopword Removal: Stopwords are common words (such as "the", "is", and "in") that appear frequently across texts but carry little meaningful information for analysis. Removing these words helps in focusing on words that have more predictive power for the sentiment analysis.
- Special Character Removal: Non-alphanumeric characters and punctuation are removed because they are generally not useful in understanding the sentiment of a text. This also helps in reducing the number of unique tokens the model needs to handle.
- Stemming and Lemmatization: These techniques reduce words to their root or base form. For instance, "stocks", "stocker", and "stocking" might all be reduced to "stock". This helps in aggregating the sentiment expressed towards similar concepts. Stemming is faster but cruder as it chops off word ends, while lemmatization is more sophisticated, using vocabulary and morphological analysis.

3.2 Feature Extraction

- Word Embeddings (Word2Vec): Instead of using traditional count-based methods like Bag-of-Words (BoW) or TF-IDF, which result in sparse and high-dimensional vectors, Word2Vec creates dense vectors for each word. These vectors capture semantic meanings, where similar words have similar encoding. Word2Vec uses neural networks to learn word associations from a large corpus of text. In this project, a Word2Vec model is trained on the preprocessed news headlines to capture the context within the financial news domain.
- Vector Averaging for Sentences: After converting words into vectors, sentence-level features are created by averaging the word vectors for each sentence (or document). This method preserves the semantic of the entire sentence and reduces the dimensionality of the data, which is beneficial for the efficiency of subsequent modeling steps.
- Dimensionality Reduction and Normalization: Additional steps can include reducing the dimensionality further using techniques like PCA (Principal Component Analysis) if necessary, and normalizing features to ensure that model weights are initialized and updated uniformly during training.

3.3 Algorithms/Strategies Used

Word2Vec for Semantic Analysis: This method is chosen for its ability to capture contextual relationships between words, making it suitable for sentiment analysis.

3.4 Design Choices and Assumptions

Vectorization: The choice to use Word2Vec over other methods like TF-IDF or direct count vectors is based on the need for dense representations that support nuanced semantic analysis.

4 MODEL DEVELOPMENT

The sentiment_model.py script develops a neural network to perform sentiment analysis based on the features extracted from the news headlines.

4.1 Neural Network Architecture

A convolutional neural network (CNN) is used for its efficacy in handling sequential input data (text).

4.2 Model Selection

The model chosen for this task is a Convolutional Neural Network (CNN). While CNNs are traditionally known for image processing tasks, they are also highly effective for sequence processing of text data due to their ability to extract higher-order features through convolutional layers.

4.3 Input Layer

The input to the model consists of fixed-size vectors obtained from the Word2Vec model. These vectors undergo preprocessing to ensure they are suitable for neural network processing, including normalization.

4.4 Convolutional Layers

The model uses 1D convolutional layers which are ideal for extracting features from sequences. In the context of text, these layers slide over word embeddings, capturing local dependencies and semantic patterns.

4.5 Pooling Layers

Following convolutional layers, max pooling layers are used to reduce the dimensionality of the data, which helps in reducing the computational load and also in extracting the most dominant features which are robust against the noise in the data.

4.6 Dropout Layers

Dropout is implemented as a regularization method to prevent overfitting. By randomly dropping units in the neural network during training, it forces the network to learn robust features that are useful in conjunction with many different random subsets of the other neurons.

4.7 Fully Connected Layers

The output from the convolutional and pooling layers is flattened and fed into fully connected (dense) layers. These layers synthesize the features extracted by the convolutions to make a final prediction regarding the sentiment.

4.8 Output Layer

The output layer consists of neurons equal to the number of sentiment classes (positive, neutral, negative). It typically uses a softmax activation function to output a probability distribution over the classes. Training Process: The model is trained using cross-entropy loss, with Adam optimizer and a learning rate scheduler to improve training dynamics.

4.9 Training and Validation

4.9.1 Loss Function. Cross-entropy loss is used, which is standard for classification tasks. It measures the performance of a classification model whose output is a probability value between 0 and 1.

4.9.2 Optimizer. The Adam optimizer is chosen for its effectiveness in handling sparse gradients and its adaptiveness in updating weights, which is beneficial for quickly converging in training deep neural networks.

4.9.3 Batch Processing and Iterations. The model is trained using mini-batch gradient descent, which balances the speed of stochastic gradient descent and the efficiency of batch gradient descent. The number of epochs and batch size are tuned based on the performance on a validation set.

4.9.4 Learning Rate Scheduling. A learning rate scheduler is employed to adjust the learning rate during training, which can help in finding a better global minimum during training.

4.9.5 Validation Strategy. The dataset is split into training and validation sets. The validation set is used to tune the hyperparameters and to prevent overfitting. Early stopping is also implemented to stop training when the validation loss ceases to decrease, thereby saving computational resources and preventing overfitting.

4.10 Algorithms/Strategies Used

- CNN for Text Classification: Chosen for its ability to extract higher-order features from text data, which is crucial in understanding the sentiment expressed in news headlines.
- Feature Representation: It is assumed that the Word2Vec embeddings provide a rich enough representation of text to capture semantic meanings necessary for sentiment analysis.
- Generalization Capability: The architecture is designed to generalize well to unseen data by employing techniques such as dropout and early stopping based on the validation set performance.

4.11 Design Choices and Assumptions

Data Splitting: The data is split into training and testing sets to evaluate the model's performance on unseen data, ensuring that it generalizes well.

5 STRATEGY SIMULATION

Simulation of trading strategies based on the output of the sentiment analysis model is conducted to test the potential efficacy of trading decisions influenced by news sentiment.

5.1 Design and Implementation of Trading Strategies

5.1.1 Strategy Design Based on Sentiment Scores. The core of the simulation involves creating rules that translate sentiment scores into actionable trading signals. For example:

Positive Sentiment. If the sentiment analysis outputs a strong positive score, the strategy might consider this an indicator to buy or hold a position, anticipating upward price movement.

Negative Sentiment. Conversely, a strong negative sentiment might trigger a sell signal or short-selling, anticipating downward movement.

Neutral Sentiment. In cases of neutral sentiment, the strategy might opt to hold its current positions or seek other indicators for decision-making.

Threshold Setting. Critical to the strategy are the thresholds set for interpreting the sentiment scores. These thresholds determine how strongly positive or negative sentiment must be to trigger buying or selling actions.

Combining Multiple Indicators. Beyond sentiment, the strategy can integrate other technical indicators (e.g., moving averages, volatility indices) to confirm signals or mitigate risks, thus creating a more robust trading system.

5.2 Simulation Environment and Backtesting

5.2.1 Simulation Tools and Platforms. The strategies are tested using a trading simulation platform that mimics the behavior of financial markets. This platform allows the integration of the trading strategy logic coded in Python, leveraging historical market data to simulate trades.

5.2.2 Data Handling. Historical market data, including price, volume, and perhaps other market indicators, are used alongside the sentiment data to test the strategy. This data must be aligned in time with the sentiment analysis outputs to ensure that the simulation reflects realistic trading scenarios.

5.2.3 Backtesting Framework. The backtesting involves running the trading strategy against historical data to evaluate its effectiveness. Key metrics such as return on investment, maximum drawdown, and the Sharpe ratio are calculated to assess performance.

5.3 Risk Management and Optimization

5.3.1 Risk Considerations. The simulation incorporates risk management protocols, such as setting stop-loss orders, diversifying holdings, and limiting exposure per trade. These measures are designed to protect against large losses in adverse market conditions.

5.3.2 Optimization Techniques. Parameters within the strategy, such as sentiment thresholds and the parameters of other technical indicators, are optimized based on historical performance. Techniques such as grid search, random search, or even machine learning methods like reinforcement learning can be employed to find the optimal settings.

5.4 Evaluation and Iterative Improvement

5.4.1 Performance Evaluation. After backtesting, the strategy's performance is analyzed in depth. This includes studying the strategy's ability to capitalize on market opportunities and its resilience during market downturns.

5.4.2 Iterative Refinement. Based on the outcomes of the initial simulations, the strategy may undergo iterative refinements to improve its performance or adapt to new market insights. This iterative process is crucial as it allows the strategy to evolve in response to feedback from its performance in simulated environments.

6 INTEGRATION WITH TWS FOR SIMULATED TRADING

The tws.py script integrates the trading strategies with the Interactive Brokers Trader Workstation (TWS) for simulated trading.

6.1 Methodology

- API Connection: Establishes a connection to TWS using the ib_insync library, which allows for programmatic control over trading actions. Order Execution: Based on the sentiment analysis, trades are executed in a simulated

- environment to test the integration and operational efficiency of the system.

6.2 Design Choices and Assumptions

Use of Simulated Environment: The use of a simulated trading environment is crucial for safely testing the system without financial risk.