Stock Price Direction Prediction Based on Deep Learning

DATS 6303 Deep Learning Final Individual Project Report

Jianjun Gao

Introduction

This project focuses on predicting stock price directions using various deep learning models. The main objective was to develop and compare different neural network architectures for forecasting whether a stock's price will rise or fall in the following year based on financial statements and historical price data. Our group project involved data collection, preprocessing, model development, and performance analysis, with my primary focus being on the data preprocessing and model implementation aspects.

Individual Work Background

My contribution centered on fetching data by using FMP API, developing the data preprocessing pipeline and implementing three key neural network architectures: LSTM, RNN, and CNN-LSTM. The theoretical foundation for this work builds on the following concepts:

Time Series Data Processing

- Feature normalization using MinMaxScaler to ensure all financial metrics are on the same scale
- Time series sequence creation with a 5-time step window
- Binary label generation for price direction prediction

Neural Network Architectures

The implemented models were chosen based on their theoretical strengths:

- LSTM: Capable of capturing long-term dependencies in sequential data
- RNN: Efficient for short-term pattern recognition
- CNN-LSTM: Combines local feature extraction with sequential learning

Detailed Work Description

Data Fetching and Combination Implementation

1. Data Collection and API Integration:

Used the Financial Modeling Prep API to fetch financial statements data. Retrieved a list of stocks that have available financial statements. Filtered to find the intersection between common stocks and stocks with statements

2. Financial Statement Processing:

Set up three types of financial statements to collect:

- Income statements
- Balance sheet statements
- Cash flow statements

Created a loop to fetch data for each stock symbol. Handled API responses and error cases properly

3. Data Consolidation:

Combined data from all three statement types for each symbol. Merged the statements using common columns (date, symbol, calendarYear, period). Concatenated data from all symbols into a single DataFrame

4. Data Cleaning and Organization:

Dropped rows with missing values. Removed unnecessary columns (like reportedCurrency, cik, fillingDate, etc.). Added company names to the financial data. Reordered columns to place 'name' next to 'symbol' for better readability

5. Data Quality Checks:

Implemented a function to check for columns containing only zeros. Printed columns that contained only zero values for review

6. Data Export:

Saved the final processed dataset in both CSV and Excel formats

- 'final_data.csv'
- 'final_data.xlsx'

Data Preprocessing Implementation

I developed the data preprocessing pipeline, which included:

1. Financial Data Cleaning:

```
df = pd.read_csv('final_data_with_adjClose.csv')
df.dropna(inplace=True)
df.reset_index(drop=True, inplace=True)
```

2. Label Generation:

```
def create_labels(group):
    group = group.sort_values('calendarYear')
    group['price_direction'] = (group['adjClose'].shift(-1) > group['adjClose']).astype(int)
    return group
df = df.groupby('symbol', group_keys=False).apply(create_labels)
```

3. Feature Engineering:

```
features = ['revenue', 'grossProfit', 'operatingIncome', 'netIncome_x',
            'cashAndCashEquivalents', 'totalAssets', 'totalLiabilities', 'adjClose']
df['eps_normalized'] = df['eps'] / df['adjClose']
df['debt_to_equity'] = df['totalDebt'] / (df['totalEquity'] + 1e-6)
df['market_cap'] = df['adjClose'] * df['weightedAverageShsOut']
```

Model Implementation

I implemented three neural network architectures:

1. LSTM Model:

```
def build_lstm_model():
    model = Sequential()
    model.add(Bidirectional(LSTM(128, return_sequences=True)))
    model.add(Dropout(0.3))
    model.add(LSTM(64, return_sequences=False))
    model.add(Dense(64, activation='relu'))
```

```
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    return model
```

2. RNN Model:

```
def build_rnn_model():
    model = Sequential()
    model.add(SimpleRNN(128, return_sequences=True))
    model.add(SimpleRNN(64, return_sequences=False))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    return model
```

3. CNN-LSTM Model:

```
def build_cnn_lstm_model():
    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Bidirectional(LSTM(64, return_sequences=False)))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    return model
```

Results

Model Performance Comparison

1. LSTM Performance:

- Achieved the best test accuracy at 64%
- Showed stable learning curves
- Demonstrated good capability in capturing long-term patterns

2. RNN Performance:

- Slightly lower performance than LSTM

- Showed more volatility in training

- Limited by vanishing gradient issues

3. CNN-LSTM Performance:

- Lowest accuracy among the three models

- Indicated that local patterns might not be as significant for this task

- Higher computational cost without corresponding performance gains

Summary and Conclusions

Key Achievements

- Successfully implemented three different neural network architectures

- Developed a robust data preprocessing pipeline

- Achieved reasonable prediction accuracy considering the complexity of stock market prediction

Lessons Learned

- LSTM architectures are more suitable for financial time series prediction

- Feature engineering significantly impacts model performance

- The importance of proper data normalization and preprocessing

Future Improvements

I will be looking at the feature selection more carefully instead of just using correlation matrix to filter out features. I will also try fetching more financial statements data for example 10 years of quarterly reports. I also want to deploy an OpenAI API therefore I can upload new financial statements and let LLM model extract the data out and put them into the model we

have trained.

What's more, we are going to make a streamlit demo for it, which we failed to make one during presentation.

Code Origin Analysis

Original Code Lines: 150

Modified Code Lines: 70

Internet-Sourced Code Lines: 129

Percentage calculation:

(129 - 70) / 279 × 100 = 21.15%

References

1. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

2. Keras Documentation. (2024). Retrieved from https://keras.io/

3. Chollet, F. (2017). Deep learning with Python. Manning Publications.

4. Financial Modeling Prep API Documentation. Retrieved from https://financialmodelingprep.com/developer/docs/

5. TensorFlow Documentation. (2024). Retrieved from https://www.tensorflow.org/api_docs