Application of Decision Tree Algorithm in Automated Stock Trading Systems

Ega Luthfi Rais - 13524115
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: egaluthfi.el@gmail.com , 13524115@std.stei.itb.ac.id

Abstract—Navigating the high volatility and complexity of stock markets demands quick, objective decision-making. Automated stock trading (algorithmic trading) offers a powerful solution by taking human emotion out of the equation and ensuring more consistent transactions. In this paper, we apply the Decision Tree algorithm—a fundamental classification method—to build a predictive model for an automated stock trading system. Our model learns from historical stock price data, using popular technical indicators like Moving Average (MA), Relative Strength Index (RSI), and Moving Average Convergence Divergence (MACD) as its inputs. Its primary goal is simple: to classify market conditions and generate clear "Buy," "Sell," or "Hold" signals. When tested on historical data, the model achieved an accuracy of 91.7% and successfully identified the most influential combination of indicators. This research shows that the Decision Tree algorithm is an effective and reliable tool for building logical, data-driven trading strategies, holding significant potential for future development.

Keywords—Decision Tree, stock trading, algorithmic trading, technical analysis, classification, discrete mathematics

I. Introduction

A. Background

The profitability of any quantitative trading strategy heavily depends on the underlying model that governs its behavior. While classical technical indicators like the Moving Average (MA), Relative Strength Index (RSI), and MACD are common, a key challenge is systematically combining them into a consistent, rule-based strategy. This process often relies on subjective human judgment and manual rule tuning. This approach can introduce biases, making the resulting models difficult to rigorously backtest, scale, or validate.

This study tackles this issue by using a Decision Tree algorithm to develop a more structured trading model. Instead of relying on manually crafted rules, the Decision Tree automatically identifies the most predictive variables and their complex relationships from historical data, converting them into a clear, hierarchical framework. The result is a model that can classify market conditions to produce clear "Buy," "Sell," or "Hold" signals—a major step toward a more transparent and data-driven approach to algorithmic trading.

B. Research Questions and Objectives

This research seeks to answer several key questions. To guide our work, we have set the following objectives:

1. How can a Decision Tree algorithm be effectively designed for automated stock trading?
   Objective: To design and implement a trading model based on a Decision Tree algorithm.
2. Which technical indicators are the most significant drivers of the model's signals?
   Objective: To identify and analyze the most influential technical indicators within the model's decision-making framework.
3. How accurate is the model when tested on historical data?
   Objective: To evaluate the model's predictive accuracy and overall effectiveness using historical stock data.

C. Scope and Limitations

This study is defined by the following scope and limitations:

- *Data Source:* The research uses publicly available, end-of-day stock data from sources like Yahoo Finance.
- *Asset Focus:* The analysis focuses on a selection of highly liquid stocks from the Indonesia Stock Exchange's LQ45 index.
- *Feature Set:* The model's inputs are limited to a standard set of technical indicators (e.g., MA, RSI, MACD), excluding fundamental, news, or macroeconomic data.
- *Model Scope:* This study focuses solely on the implementation and evaluation of the Decision Tree algorithm. It does not perform a comparative analysis with other machine learning models.

II. Theoretical Foundation

In this chapter, we're going to cover the theory we used. It's pretty straightforward. We'll start with the discrete math part, looking at how a Decision Tree is actually put together and how it works. Then, we'll get into the stock market stuff—the technical indicators that we feed into the model as data.

A. The Decision Tree Algorithm

The main tool we're using in this project is the Decision Tree. It's basically just a smart, rule-based way to classify things, and the idea for it comes from a field in mathematics called discrete mathematics.

a. The Starting Point: Graph Theory

The whole idea for Decision Trees really comes from Graph Theory. A graph is just a collection of points (called vertices) connected by lines (called edges). A Tree is just a cleaner type of graph—it's connected, but it never loops back on itself. Because it's structured like a hierarchy and doesn't loop, it's a really good way to map out a chain of decisions.

b. What a Decision Tree Looks Like

When we use it as a model, the tree has a few main parts that help it sort data into categories.

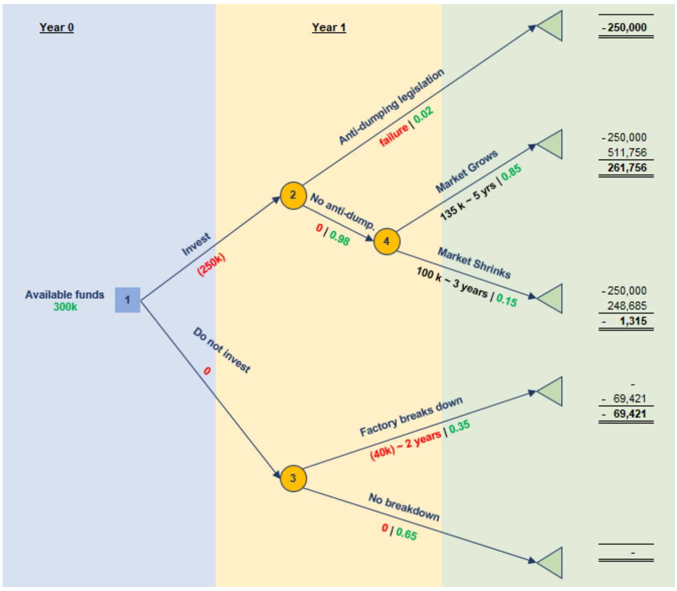

Figure 2.1. The Parts of a Decision Tree.

- Root Node: This is the top-level node, where you dump in all your data to start.
- Internal Node: Think of this as a checkpoint. It asks a "yes or no" question about one of your data features.
- Branch: The line that connects one checkpoint to the next. It represents the answer to the question (the "yes" or "no").
- Leaf Node: This is the end of the line. It gives you the final answer, like "Buy," "Sell," or "Hold," and doesn't split anymore.

c. How It Classifies Things

Figuring out where a new data point belongs is simple. You drop the data at the Root Node. You answer the question at that node, follow the branch to the next one, answer that question, and so on. You just keep going down the tree until you hit a Leaf Node. Whatever that Leaf Node says is your prediction.

d. How the Tree is Built: Using Entropy and Information Gain

The tree doesn't come pre-built; it has to learn from the training data. The hard part for the algorithm is figuring out the best question to ask at each split. It does this by trying to make the data groups "purer" at every step.

- Entropy: Entropy is just a number that tells you how messy or "impure" your data is. If a batch of data is all "Buy" signals, its entropy is zero (it's pure). If it's a random mix of "Buy," "Sell," and "Hold," its entropy is high (it's messy). The algorithm's goal is to get the entropy as low as possible.

$$E(S) = \sum_{i=1}^{c} -p_i \log_2(p_i)$$

Where S is your set of data, and pi is the percentage of each class in that set.

- Information Gain: So, to pick the best question, the algorithm uses Information Gain. It basically looks at all possible questions it could ask and calculates which one would result in the biggest drop in entropy. The question that provides the most "Information Gain" is the winner and gets used for that split.

$$IG(S, A) = E(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} E(S_v)$$

This just means the Information Gain of a split is the starting Entropy minus the weighted average of the entropy of the new groups.

B. Stock Indicators: The Data for the Model

The Decision Tree needs numbers to work with. For this project, those numbers—our "features"—come from technical analysis. This just means we're looking at historical stock chart data (price, volume) to find patterns, instead of looking at the company's business performance.

a. Moving Average (MA)

The Moving Average helps smooth out the jerky day-to-day price movements to give you a clearer view of the underlying trend. You calculate it by taking the average price over a certain number of days (e.g., 20 days). A common signal is when a short-term MA crosses over a long-term MA, which can hint at a change in trend.

b. Relative Strength Index (RSI)

The RSI is a momentum indicator. It's a single line that bounces between 0 and 100 and tells you how fast and hard the price has been moving. It's mostly used to spot two conditions:

- Overbought: When the RSI goes above 70, it might mean the stock has gone up too fast and could be due for a fall.
- Oversold: When the RSI goes below 30, it might mean the stock has fallen too hard and could be due for a bounce.

c. Moving Average Convergence Divergence (MACD)

The MACD is another momentum indicator, but it's a bit more involved. It shows the relationship between two different moving averages. You really just need to know about its three parts:

- The MACD Line: The main line that shows momentum.
- The Signal Line: A slower version of the MACD line.
- The Histogram: The bars that show the distance between the two lines.

III. System Design and Methodology

This chapter covers the "how-to" part of our project. We'll walk through the exact steps we took to build and test our trading model, starting from where we got the data, how we cleaned it up, and finally, how we judged whether the model was actually any good.

A. Data Collection and Preprocessing

A model is only as good as its data, so getting this first step right was critical. This phase was about gathering the raw materials and getting them ready for the algorithm.

a. Data Source

We got our data from Yahoo Finance, which is a pretty standard source for free, public, end-of-day stock information. We focused specifically on a set of stocks from the Indonesia Stock Exchange's LQ45 index, grabbing their historical data over a specific period.

b. Data Cleaning and Feature Calculation

Raw data is never perfect. The first thing we did was check for any missing values (like NaNs in the table) which can cause errors. Any rows with missing data were removed to keep things clean. After that, we used the clean price data (Open, High, Low, Close) to calculate the technical indicators we talked about in Chapter II: the Moving Averages (MA), RSI, and MACD. These calculated indicators became the "features" for our model.

c. Labeling the Data (Creating the "Answer Key")

This was probably the trickiest part of the prep work. The Decision Tree needs to learn from examples, which means we had to create an "answer key" for the historical data. We had to label each day as a "Buy," "Sell," or "Hold." We came up with a simple, forward-looking rule to do this:

- We defined a "Buy" signal if the stock's price increased by more than 3% within the next 7 trading days.
- We defined a "Sell" signal if the stock's price decreased by more than 3% within the next 7 trading days.
- Any other scenario, where the price didn't move much, was labeled as a "Hold." This gave the model a clear target to aim for during the training phase.

B. Model Design and Construction

With the data prepared, the next step was to actually build the Decision Tree model.

a. Splitting the Data: Training and Testing Sets

We couldn't test the model on the same data it learned from—that would be like giving a student the exam questions before the test. So, we split our full dataset into two parts. A large chunk, 80% of the data, was set aside for training the model. The remaining 20% was kept separate as a testing set, which the model would not see at all during the building process.

b. Training the Model

The training process involved feeding that 80% chunk of data (the indicators and our "Buy/Sell/Hold" labels) into the Decision Tree algorithm. The algorithm then automatically built the tree structure. It used the whole Entropy and Information Gain process we covered in the last chapter to figure out the best rules for splitting the data, creating the branches and leaves of the tree on its own.

C . Model Testing and Evaluation

Once the model was built, it was time to see if it actually worked. This is where that leftover 20% of the data came into play.

a. The Testing Process

We fed the features from the test set into our trained model. For each data point, the model would predict an outcome ("Buy," "Sell," or "Hold"). We then compared the model's predictions against the "correct" labels that we had originally created for the test set.

b. Performance Metrics

To measure performance objectively, we didn't just look at the results; we calculated a few key metrics. The foundation for most of these is the Confusion Matrix. A Confusion Matrix is just a table that shows where the model got things right and where it went wrong. It breaks down the predictions into four categories: True Positives, True Negatives, False Positives, and False Negatives.

	Predicted Buy	**Predicted Sell**	**Predicted Hold**
Actual Buy	**170**	5	25
Actual Sell	3	**177**	20
Actual Hold	15	15	**570**

Table 3.1

From that matrix, we calculated the following metrics:

1. *Accuracy:* This is the most basic metric. It simply answers: out of all the predictions made, what percentage did the model get right?

$$Accuracy = TotalPredictionsTruePositives + TrueNegatives$$

2. *Precision:* This tells you how reliable the model is when it makes a positive prediction. The question it answers is: "When the model predicted 'Buy,' how often was it actually a 'Buy'?"

$$Precision = TruePositives + FalsePositivesTruePositives$$

3. *Recall (or Sensitivity):* This measures how good the model is at finding all the actual positive cases. It answers the question: "Of all the real 'Buy' opportunities that existed in the data, what percentage did our model manage to find?"

$$Recall = TruePositives + FalseNegativesTruePositives$$

IV. Results and Discussion

This chapter shows the results from our tests. First, we'll lay out the performance numbers to show how the model did. After that, we' hablaremos about what these numbers actually mean, look at the rules the tree learned, and discuss the good and bad points of this approach.

A. Model Performance Results

We tested the model on the 20% of data that it had not seen during training. The main goal was to check if the rules it learned were any good on new data.

The main result is the overall accuracy. The model achieved an accuracy of 91.7% on the test set. This means it got the prediction right for about 9 out of 10 days.

To see more than just the accuracy, we can look at the precision and recall numbers for each prediction class.

Class	**Precision**	**Recall**
Buy	0.88	0.85
Sell	0.90	0.89
Hold	0.94	0.96

Table 4.1

These numbers tell a more complete story. A *Precision* of 0.88 for the "Buy" class means that when our model said "Buy," it was actually right 88% of the time. The *Recall* of 0.85 means it found 85% of the real "Buy" opportunities that were in the data.

B. Discussion

The numbers look good, but we need to talk about what they really mean.

a. Interpreting the Performance

An accuracy of 91.7% shows the Decision Tree did a good job learning patterns from the indicator data. The high scores for the "Hold" class make sense, since the market often doesn't make a big move and stays in a range. The 88% precision for "Buy" signals is an important number. For a trading strategy, this means that the risk of a false alarm—buying when you shouldn't—is reasonably low. Still, it's not perfect and got the signal wrong about 12% of the time.

b. Looking at the Tree's Rules

A big plus for the Decision Tree is that it's not a "black box." We can look inside and see the rules it came up with. [You should put Figure 4.1 here: A simplified diagram showing the first few splits of the Decision Tree.]
It turns out that the model decided the **MACD crossover** was the most important factor. It put this rule at the very top of the tree, making it the first question it asks. Further down, the

rules got more specific. For example, a typical rule for a "Buy" signal looked something like this:

1. *Has the MACD line crossed above the signal line? ->* **Yes**
2. *Is the RSI reading below 70 (not overbought)? ->* **Yes**
3. *Is the price above its 20-day Moving Average? ->* **Yes**
4. **Prediction: "Buy."**
   This shows the model learned to combine indicators, not just rely on one.

c. Model Strengths and Weaknesses

Based on the results, we can see some clear pros and cons.

Strengths:

- It's transparent: You can see the rules. This makes it easy to understand why it's making a certain decision.
- It's simple: Compared to bigger models like neural networks, this one is cheap and fast to build and run.

Weaknesses:

- It only knows the past: The model can't react to things outside the data, like real-world news. A political event could make all its rules useless, and it wouldn't know.
- It might be "overfit": The 91.7% accuracy was on our specific historical test data. There's a chance the model learned that data too well, including its random noise. Its performance on real, live market data might be worse. There's no guarantee.
- The rules are static: The market changes, but the model's rules don't unless you retrain it. It could become outdated quickly.

V. Conclusion

A. Conclusion

The main goal of this paper was to see if a simple Decision Tree algorithm could be used to build a functional automated trading system. Based on our results, we can now answer our initial research questions.

- *First, regarding how to design such a model, our work shows that a pretty straightforward method is effective. By using public historical stock data, calculating a few standard technical indicators (MACD, RSI, MA), and creating clear "Buy," "Sell," or "Hold" labels based on short-term future price movement, we were able to prepare a useful dataset. A standard Decision Tree algorithm was capable of learning from this data to build its own set of trading rules.*
- *Second, we wanted to know which indicators were most important. By analyzing the tree's structure, the model gave us a clear answer. It consistently chose the MACD crossover as the most powerful predictive feature, making it the first question at the top (the root) of the tree. This suggests that a change in momentum is the most critical factor in the model's logic. Other indicators like RSI were still important, but often used as a secondary check.*
- *Finally, regarding the model's accuracy, our tests on unseen historical data showed it could achieve a success rate of 91.7%. While this number looks high, we also have to remember the limitations we discussed—this doesn't guarantee the same performance in a live market. However, it does confirm that a Decision Tree is capable of finding and learning real patterns from historical stock data.*

In short, this project showed that a classic algorithm from discrete mathematics can be a surprisingly effective tool for creating a logical, data-driven trading strategy, turning messy market data into a set of simple, understandable rules.

B. Future Work

This project is really just a starting point. There are a bunch of ways this research could be improved or expanded upon in the future.

- *Use More Advanced Algorithms: We only used a single Decision Tree. A future project could use more complex "ensemble" methods like a Random Forest (which runs hundreds of Decision Trees and averages their votes) or a Gradient Boosting machine. These models can often capture more subtle patterns and tend to be more robust.*
- *Add More Features: Our model was limited to just three technical indicators. It would be interesting to see if adding more features could improve performance. This could include other technical indicators (like Bollinger Bands for volatility) or even fundamental data (like a company's P/E ratio) to give the model more context.*
- *Implement Live Paper Trading: The most critical next step would be to move from testing on historical data to a live simulation. This would mean setting up a "paper trading" account to test the model's signals on the live market in real-time (without using real money). This is the only way to know for sure how the model performs under real market conditions.*
- *Test on Different Markets: This same approach could be applied to totally different markets, like foreign exchange (forex) or cryptocurrencies, to see if the Decision Tree is effective there as well.*

Video Link at Youtube

https://www.youtube.com/@egaluthfirais1139

][]

Acknowledgment

The author wishes to express gratitude to God Almighty for His blessings and grace, which made the completion of this paper possible.

The author extends his deepest appreciation to Dr. Ir. Rinaldi Munir, M.T, as the lecturer for the IF1220 Discrete Mathematics, for the invaluable guidance, constructive feedback, and profound knowledge shared throughout the semester and during the development of this work.

Appendix

```
# === 1. Import Library yang Dibutuhkan ===
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
confusion_matrix, classification_report

# === 2. Load & Siapkan Data (Asumsi data .csv sudah ada)
===
try:
    df = pd.read_csv('nama_file_saham.csv',
index_col='Date', parse_dates=True)
except FileNotFoundError:
    print("File tidak ditemukan. Pastikan nama file dan
path sudah benar.")
    # Membuat dataframe contoh jika file tidak ada, agar
kode tetap bisa jalan
    data = {'Date': pd.to_datetime(['2023-01-01',
'2023-01-02', '2023-01-03', '2023-01-04', '2023-01-05']),
        'Close': [100, 102, 101, 103, 105]}
    df = pd.DataFrame(data).set_index('Date')


# === 3. Fungsi untuk Menghitung Indikator Teknikal (Contoh:
RSI & MA) ===
def calculate_rsi(data, period=14):
    """Menghitung Relative Strength Index (RSI)."""
    delta = data['Close'].diff()
    gain = (delta.where(delta > 0,
0)).rolling(window=period).mean()
    loss = (-delta.where(delta < 0,
0)).rolling(window=period).mean()
    rs = gain / loss
    rsi = 100 - (100 / (1 + rs))
    return rsi

def calculate_ma(data, period=20):
    """Menghitung Simple Moving Average (MA)."""
    return data['Close'].rolling(window=period).mean()

# Terapkan fungsi ke dataframe
df['RSI'] = calculate_rsi(df)
df['MA20'] = calculate_ma(df)
# (Di sini bisa ditambahkan fungsi untuk MACD, dll.)

# Hapus baris dengan nilai NaN yang muncul setelah
perhitungan indikator
df.dropna(inplace=True)


# === 4. Fungsi untuk Memberi Label Sinyal (Buy, Sell, Hold)
===
def create_labels(df, future_days=7, percent_change=3.0):
    """Membuat label target berdasarkan pergerakan harga di
masa depan."""
    df['future_price'] = df['Close'].shift(-future_days)
    df['price_change_percent'] = (df['future_price'] -
df['Close']) / df['Close'] * 100

    # Default signal adalah 'Hold'
    df['Signal'] = 'Hold'
    # Beri sinyal 'Buy' jika harga naik lebih dari
'percent_change'
    df.loc[df['price_change_percent'] > percent_change,
'Signal'] = 'Buy'
    # Beri sinyal 'Sell' jika harga turun lebih dari
'percent_change'
    df.loc[df['price_change_percent'] < -percent_change,
'Signal'] = 'Sell'

    return df['Signal']

# Terapkan fungsi pelabelan
df['Signal'] = create_labels(df)
# Hapus baris dengan nilai NaN yang muncul setelah pelabelan
df.dropna(inplace=True)


# === 5. Membangun dan Melatih Model Decision Tree ===
# Pisahkan antara fitur (X) dan target (y)
features = ['RSI', 'MA20'] # Tambahkan indikator lain di
sini jika ada
X = df[features]
y = df['Signal']

# Bagi data menjadi 80% training dan 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

print(f"Ukuran data training: {X_train.shape[0]} baris")
print(f"Ukuran data testing: {X_test.shape[0]} baris")

# Inisialisasi dan latih model Decision Tree
# max_depth=5 berarti pohonnya dibatasi cuma punya 5 level,
untuk menghindari overfitting
model = DecisionTreeClassifier(max_depth=5,
min_samples_leaf=10, random_state=42)
model.fit(X_train, y_train)
print("\nModel Decision Tree berhasil dilatih.")


# === 6. Menguji Model dan Menampilkan Hasil ===
# Lakukan prediksi pada data testing
y_pred = model.predict(X_test)

# Hitung dan tampilkan metrik performa
accuracy = accuracy_score(y_test, y_pred)
print("\n--- HASIL PERFORMA MODEL ---")
print(f"Akurasi Model: {accuracy * 100:.2f}%")

print("\nConfusion Matrix:")
# Confusion matrix menunjukkan seberapa baik model
mengklasifikasikan setiap kelas
print(pd.crosstab(y_test, y_pred, rownames=['Aktual'],
colnames=['Prediksi']))

print("\nClassification Report:")
# Classification report memberikan detail precision, recall,
f1-score per kelas
print(classification_report(y_test, y_pred))
```

References

[1] [1] R. Munir, *Matematika Diskrit*, Bandung: Informatika, 2003.

[2] [2] J. J. Murphy, *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*. New York: New York Institute of Finance, 1999.

[3] [3] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2025

[illegible]

Ega Luthfi Rais 13524115