Stock Price Prediction Using Machine Learning in Python

Abstract

Stock market prediction has always been a challenging task due to the highly volatile and non-linear nature of financial data. This project focuses on predicting a **buy or not-buy signal** for Tesla stock using Machine Learning techniques rather than predicting the exact stock price. Historical stock data from 2010 to 2017 has been analyzed, relevant features have been engineered, and multiple classification models have been trained and evaluated. The study concludes that basic machine learning models with limited features are insufficient for accurate stock market prediction, highlighting the complexity of financial markets.

---

1. Introduction

Machine Learning (ML) has found wide applications in finance, particularly in algorithmic trading and stock market analysis. Instead of forecasting precise prices, which is extremely difficult, this project aims to predict whether buying a stock on a given day will be profitable or not.

The project uses historical **OHLC (Open, High, Low, Close)** data of Tesla stock to generate a binary classification problem: - **1 (Buy)** → If the next day's closing price is higher than today's closing price - **0 (Do Not Buy)** → Otherwise

---

2. Tools and Technologies Used

- **Programming Language:** Python
- **Libraries:** NumPy, Pandas, Matplotlib, Seaborn
- **Machine Learning Framework:** Scikit-learn, XGBoost
- **Platform:** Google Colab / Jupyter Notebook

---

3. Dataset Description

The dataset consists of Tesla stock price data from **1st January 2010 to 31st December 2017**.

Attributes:

- Date
- Open
- High
- Low
- Close
- Adj Close
- Volume

Total records: **1692 rows**

The `Adj Close` column was removed as it was identical to the `Close` column.

---

4. Importing Required Libraries (Code Explanation)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
```

- NumPy: Numerical computations - Pandas: Data manipulation and analysis - Matplotlib & Seaborn: Data visualization

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn import metrics
```

- Used for preprocessing, model training, and evaluation

```
import warnings
warnings.filterwarnings('ignore')
```

- Suppresses unnecessary warning messages

---

5. Data Loading

```
df = pd.read_csv('Tesla.csv')
```

- Loads Tesla stock dataset into a DataFrame

```
df.shape
```

- Displays number of rows and columns in the dataset

---

6. Exploratory Data Analysis (EDA)

Stock Price Trend

```
plt.figure(figsize=(15,5))
plt.plot(df['Close'])
plt.title('Tesla Close Price')
plt.show()
```

- Shows an upward trend in Tesla stock prices over time

Removing Redundant Column

```
df = df.drop(['Adj Close'], axis=1)
```

- Removes redundant data

Checking Missing Values

```
df.isnull().sum()
```

- Confirms no missing values in the dataset

---

7. Feature Engineering

Extracting Date Features

```
splitted = df['Date'].str.split('/', expand=True)

df['day'] = splitted[1].astype(int)
df['month'] = splitted[0].astype(int)
df['year'] = splitted[2].astype(int)
```

- Extracts day, month, and year from Date column

Quarter-End Feature

```
df['is_quarter_end'] = np.where(df['month'] % 3 == 0, 1, 0)
```

- Identifies quarter-end months which impact stock prices

Price-Based Features

```
df['open-close'] = df['Open'] - df['Close']
df['low-high'] = df['Low'] - df['High']
```

- Captures daily price movement behavior

Target Variable

```
df['target'] = np.where(df['Close'].shift(-1) > df['Close'], 1, 0)
```

- Creates classification label (Buy / Not Buy)

---

8. Feature Selection and Normalization

```
features = df[['open-close', 'low-high', 'is_quarter_end']]
target = df['target']
```

- Selects features with low correlation

```
scaler = StandardScaler()
features = scaler.fit_transform(features)
```

- Normalizes feature values for faster convergence

---

9. Data Splitting

```
X_train, X_valid, Y_train, Y_valid = train_test_split(
    features, target, test_size=0.1, random_state=2022)
```

- Splits dataset into training (90%) and validation (10%)

---

10. Model Training and Evaluation

Models Used

```
models = [
    LogisticRegression(),
    SVC(kernel='poly', probability=True),
```

```
    XGBClassifier()
]
```

Training and Evaluation

```
for model in models:
    model.fit(X_train, Y_train)
    print(metrics.roc_auc_score(Y_valid, model.predict_proba(X_valid)[:,1]))
```

- Trains models and evaluates using ROC-AUC score

---

11. Results and Analysis

- Logistic Regression: Stable but low accuracy
- SVM: Underperformed
- XGBoost: High training accuracy but overfitting

Overall performance was close to **50%**, similar to random guessing.

---

12. Conclusion

This project demonstrates that predicting stock market movements using basic machine learning models and limited historical price data is extremely challenging. While advanced models like XGBoost show better learning capability, they tend to overfit. Accurate stock prediction requires richer datasets, technical indicators, and advanced time-series or deep learning models.

---

13. Future Scope

- Use LSTM / GRU neural networks
- Add technical indicators (RSI, MACD)
- Incorporate news sentiment analysis
- Apply time-series cross-validation

---

14. References

- Scikit-learn Documentation
- XGBoost Documentation
- Yahoo Finance Dataset