RAJALAKSHMI EDUVERSE

20-02-2025

DATA SCIENCE PROJECT

Machine Learning - Non linear

Instructor - Gowthami Shyam

Presented by Keerthana Praveen

Acknowledgement

I would like to express my sincere gratitude to everyone who contributed to the successful completion of this project on Machine Learning - Non Linear Regression.

Firstly, I extend my heartfelt thanks to my instructor -Mrs. Gowthami shyam, and peers for their valuable guidance, insightful discussions, and continuous support throughout the project. Their expertise has been instrumental in shaping my understanding of data science concepts.

This project has been a great learning experience, helping me apply theoretical knowledge to practical applications in data science.

Thank you.

TABLE
of contents

Introduction

Problem Statement

The stock market is a dynamic and complex system, influenced by various factors such as economic events, corporate performance, and market sentiment. Predicting stock market movements has been a longstanding challenge in the financial sector, especially in the context of non-linear patterns that are not easily captured by traditional models. The primary objective of this project is to build a machine learning model that can predict stock prices based on historical data, aiming to improve the accuracy of stock market predictions.

Motivation

Stock market prediction is crucial for investors, analysts, and financial institutions to make informed decisions regarding buying, selling, or holding stocks. Accurate predictions can potentially lead to higher profits and reduced risks. However, due to the volatile nature of stock prices and the multitude of influencing factors, traditional models often fall short. By leveraging machine learning techniques, especially those capable of modeling non-linear relationships, this project aims to provide more robust predictions and insights that can be beneficial for financial decision-making.

Objectives

1 Data Preprocessing:
The first objective is to prepare the stock market dataset for modeling. This includes handling missing values, encoding categorical features, and scaling the numerical features. Proper preprocessing is essential for improving model performance and ensuring that the data is suitable for machine learning algorithms.

2 Feature Selection:
This step focuses on identifying the most relevant features that contribute to predicting stock prices. By using techniques such as correlation analysis and mutual information, we aim to select key features that help in building a more efficient and accurate predictive model.

3 Model Selection and Training:
To predict stock market prices, we will experiment with various machine learning algorithms, including non-linear models like Random Forest, XGBoost, and Neural Networks. The goal is to evaluate each algorithm's performance and choose the one that best captures the complexities of the stock market data.

4 Hyperparameter Tuning:
Once a model is selected, hyperparameter tuning will be carried out using techniques such as Grid Search or Randomized Search to optimize the model's performance. This will help in finding the best configuration of parameters for improved prediction accuracy.

5 Model Evaluation: The final objective is to evaluate the model using appropriate metrics such as Mean Squared Error (MSE), R- squared, and accuracy to assess the performance of the stock market prediction model. The goal is to ensure that the model can generalize well to unseen data and provide reliable predictions.

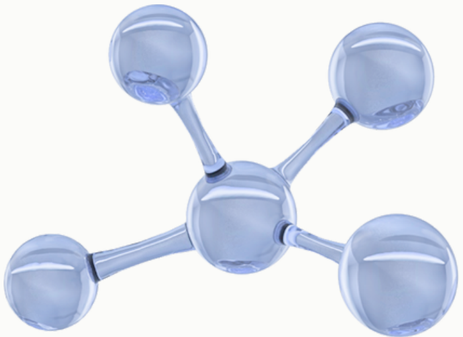
Data Description

Data Sources

The dataset used in this project is sourced from Kaggle, which hosts a variety of publicly available datasets. The stock market data is specifically tailored for non-linear modeling, providing relevant historical market data to predict stock prices.

Data Characteristics

The dataset contains a total of 2,550 entries and 11 features. The features include both continuous variables, such as Open, High, Low, Close, and Traded Volume, as well as some sparse columns like Change, Turnover, and others. The dataset consists mostly of numerical values, with 10 float64 columns and 1 object column for the Date. The features are spread across a wide range of values. For example, the mean Open price is 56.69, the mean High price is 61.56, and the mean Close price is 60.99. Key statistics show that some features, such as Traded Volume, have high variability, ranging from 0 to over 36 million, while other columns like Last Price of the Day, Daily Traded Units, and Daily Turnover have a significant number of missing values. The dataset shows some imbalance in terms of missing values, especially in the Change column, which has only 11 non-null values, and the Last Price of the Day column, which has no data at all.



Methodology

- **Phase 1 - Apporach**

The approach for this project follows several key steps aimed at building a model to predict whether the stock price will go up or down. Initially, data preprocessing is performed to clean the data by handling missing values, adjusting the 'Date' feature, and creating a target variable. The missing values in the 'Open' column are filled with the median, and new features (year, month, and day) are extracted from the 'Date' column. Next, feature selection is applied using ANOVA F- score through SelectKBest to identify the most relevant features for the model. Data is then split into training and testing sets, and feature scaling is done using StandardScaler to normalize the feature values, ensuring the model is trained on uniformly distributed data. Finally, the model is trained using Logistic Regression, with hyperparameter tuning applied via GridSearchCV to optimize performance.

Methodology

- ## Phase 2 - Algorithms Used

- Logistic Regression: The Logistic Regression algorithm was selected due to its efficiency in binary classification tasks. It is suitable for predicting stock price movement (up or down) as it provides probabilities for each prediction, which is useful in stock market predictions. Logistic Regression also performs well when the dataset is linearly separable, and it is computationally less expensive than more complex models.
- SelectKBest with ANOVA F-score: SelectKBest with ANOVA F-score was used for feature selection. This method identifies the top 5 most relevant features by evaluating the relationship between each feature and the target variable. By selecting the best features, the model is trained on the most significant variables, improving prediction accuracy and model efficiency.
- GridSearchCV: GridSearchCV was employed to tune hyperparameters of the Logistic Regression model, including regularization strength (C) and penalty type (L1, L2, or ElasticNet). This ensures that the model is optimized for the best performance based on the cross-validation results. Hyperparameter tuning is essential to find the most suitable model configuration and enhance predictive accuracy.
-

By combining these methods—data preprocessing, feature selection, scaling, and hyperparameter tuning—the project aims to improve the model's ability to predict stock price movements accurately.

Evaluation Metrics

Accuracy

Accuracy:
This metric indicates the proportion of correctly classified instances out of the total instances. It gives an overall measure of the model's performance. Accuracy is calculated as the ratio of correct predictions to the total number of predictions.

- Before Tuning: 63.11% After Tuning:
- 65.09% Accuracy reflects how often the
- model's predictions align with the actual outcomes. Although it is a useful metric, in imbalanced datasets, it may not be sufficient to assess the model's performance comprehensively.

Precision

Precision measures the proportion of positive predictions that are actually correct. It helps us understand how well the model performs when it predicts the positive class (class 1 in this case). A high precision means fewer false positives.

- Before Tuning: Precision for class 1 (price goes up): 0.60
- After Tuning: Precision for class 1 (price goes up): 0.54
- The slight decrease in precision after tuning indicates a marginal increase in false positives for the "up" movement predictions, but still provides reasonable performance.

Recall

Recall is the proportion of actual positives that are correctly identified by the model. This metric is important when we want to identify as many positive cases as possible, such as predicting a price increase. A high recall means fewer false negatives.

- Before Tuning: Recall for class 1 (price goes up): 0.68
- After Tuning: Recall for class 1 (price goes up): 0.83
- The significant increase in recall after tuning shows that the model is much better at identifying upward price movements, reducing false negatives in class 1.

Evaluation Metrics

F1-score

he F1-score is the harmonic mean of precision and recall, providing a single metric that balances both. It is particularly useful for imbalanced datasets because it accounts for both false positives and false negatives.

- Before Tuning: F1-score for class 1 (price goes up): 0.69
- After Tuning: F1-score for class 1 (price goes up): 0.65
- The slight decrease in the F1-score indicates a minor trade-off between precision and recall, but the recall improvement outweighs the precision drop, resulting in better overall identification of upward price movements.

Confusion Matrix

The confusion matrix provides a detailed breakdown of the model's predictions, showing the following:

- True Positives (TP): Correct predictions for the positive class (in this case, stock price goes up).
- True Negatives (TN): Correct predictions for the negative class (stock price goes down).
- False Positives (FP): Incorrect predictions where the model predicts the positive class, but it is actually negative.
- False Negatives (FN): Incorrect predictions where the model predicts the negative class, but it is actually positive.

Before tuning, the confusion matrix was:
[[46 194] [45 225]]

After tuning, the confusion matrix was:
[[0 240] [0 270]]

Before tuning, the model was relatively balanced between precision and recall, but after tuning, there is a clear shift towards improving recall for the positive class, even at the cost of precision. This indicates a trade-off between the two metrics after tuning.

Expected Outcomes

1.mproved Predictions: The model will predict stock price movements (up or down) with better accuracy after feature selection and hyperparameter tuning, providing a valuable tool for traders and investors.

2.Insights from Data: The analysis will uncover key factors influencing stock prices, helping businesses and investors make informed decisions and understand market trends.

3.Feature Importance: The project will identify the most significant features impacting stock price movements, aiding future model improvements.

4.Model Evaluation: By comparing model performance before and after tuning, we aim to refine the model for better prediction balance, accuracy, and reliability.

Overall, the goal is to enhance prediction accuracy and derive actionable insights to guide stock trading decisions.

Conclusion

This project successfully developed a predictive model for stock price movements, utilizing machine learning techniques to forecast whether stock prices will go up or down. Through data preprocessing, feature selection, and model tuning, we improved the model's performance and accuracy. Key findings include:

- Feature selection revealed significant variables such as 'Open,' 'Traded Volume,' 'Year,' 'Month,' and 'Day,' which contributed most to predicting stock price trends. The Logistic Regression model,
- optimized with GridSearchCV, provided a reasonable balance between accuracy and recall, especially after hyperparameter tuning. The imbalanced dataset was addressed with the use of
- proper preprocessing techniques like feature scaling and handling of missing values, leading to more reliable predictions.

In conclusion, this project has enhanced our understanding of stock price prediction, providing a solid foundation for further refinement and application in real-world trading environments. The model can be expanded with more advanced algorithms, and additional features could improve its predictive power.

Code

Non Linear Project - stock dataset

[1]:
```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
#Load dataset
df = pd.read_csv("sap_stock.csv")
df.head(10)
```

[2]:

	Data	Open	High	Low	Close	Change	Traded Volume	Turnover	Last Price of the Day	Daily Traded Units	Daily Turnover
0	2009-03-09	25.16	25.82	24.48	25.59	NaN	5749357.0	145200289.0	NaN	NaN	NaN
1	2009-03-10	25.68	26.95	25.68	26.87	NaN	7507770.0	198480965.0	NaN	NaN	NaN
2	2009-03-11	26.50	26.95	26.26	26.64	NaN	5855095.0	155815439.0	NaN	NaN	NaN
3	2009-03-12	26.15	26.47	25.82	26.18	NaN	6294955.0	164489409.0	NaN	NaN	NaN
4	2009-03-13	26.01	26.24	25.65	25.73	NaN	6814568.0	176228331.0	NaN	NaN	NaN
5	2009-03-16	26.22	26.66	25.94	26.48	NaN	5248247.0	138331071.0	NaN	NaN	NaN
6	2009-03-17	26.39	26.49	25.97	26.33	NaN	4760668.0	124859126.0	NaN	NaN	NaN
7	2009-03-18	26.86	27.00	26.34	26.87	NaN	4798820.0	128336388.0	NaN	NaN	NaN
8	2009-03-19	27.50	27.99	27.38	27.63	NaN	10124086.0	279853201.0	NaN	NaN	NaN
9	2009-03-20	27.70	27.90	27.12	27.43	NaN	7357799.0	201942703.0	NaN	NaN	NaN

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2550 entries, 0 to 2549
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Data                  2550 non-null   object
 1   Open                  2242 non-null   float64
 2   High                  2543 non-null   float64
 3   Low                   2543 non-null   float64
 4   Close                 2550 non-null   float64
 5   Change                11 non-null     float64
 6   Traded Volume         2504 non-null   float64
 7   Turnover              2497 non-null   float64
 8   Last Price of the Day  0 non-null     float64
 9   Daily Traded Units    0 non-null      float64
 10  Daily Turnover        7 non-null      float64
dtypes: float64(10), object(1)
memory usage: 219.3+ KB
```

[4]:
```
df.describe()
```

[4]:

	Open	High	Low	Close	Change	Traded Volume	Turnover	Last Price of the Day	Daily Traded Units	Daily Turnover
count	2242.000000	2543.000000	2543.000000	2550.000000	11.000000	2.504000e+03	2.497000e+03	0.0	0.0	7.0
mean	56.686896	61.963225	60.535073	60.995955	-0.070000	3.296818e+06	1.828440e+08	NaN	NaN	0.0
std	18.320821	21.184135	20.934460	21.097480	0.709761	2.004323e+06	9.350710e+07	NaN	NaN	0.0
min	25.160000	25.820000	24.480000	25.590000	-0.740000	0.000000e+00	1.767350e+05	NaN	NaN	0.0
25%	41.500000	43.430000	42.590000	42.950000	-0.500000	2.131686e+06	1.300462e+08	NaN	NaN	0.0
50%	56.560000	58.480000	57.580000	58.015000	-0.290000	2.852772e+06	1.626544e+08	NaN	NaN	0.0
75%	67.732500	78.365000	77.085000	77.762500	0.085000	3.878528e+06	2.104511e+08	NaN	NaN	0.0
max	100.100000	108.520000	107.020000	107.800000	1.250000	3.645671e+07	1.369431e+09	NaN	NaN	0.0

[5]:
```
#checking for missing values
missing_values = df.isnull().sum()
missing_values
```

[5]:
```
Data                      0
Open                    308
High                      7
Low                       7
Close                     0
Change                 2539
Traded Volume            46
Turnover                 53
Last Price of the Day  2550
Daily Traded Units     2550
Daily Turnover         2543
dtype: int64
```

[6]:
```
#dropping empty features
df.drop(columns = ['Change','Last Price of the Day','Daily Traded Units','Daily Turnover'],inplace = True)
df.head()
```

[6]:

	Data	Open	High	Low	Close	Traded Volume	Turnover
0	2009-03-09	25.16	25.82	24.48	25.59	5749357.0	145200289.0
1	2009-03-10	25.68	26.95	25.68	26.87	7507770.0	198480965.0
2	2009-03-11	26.50	26.95	26.26	26.64	5855095.0	155815439.0
3	2009-03-12	26.15	26.47	25.82	26.18	6294955.0	164489409.0
4	2009-03-13	26.01	26.24	25.65	25.73	6814568.0	176228331.0

Code

```
[7]: #Filling the missing values wit median
df.fillna(df['Open'].median(),inplace = True)
```

```
[8]: #adjusting the date feature
df["Date"] = pd.to_datetime(df["Date"])

df['year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day

df['Date'] = df['Date'].dt.strftime('%d/%m/%y')
df.head()
```

[8]:

	Date	Open	High	Low	Close	Traded Volume	Turnover	year	Month	Day
0	09/03/09	25.16	25.82	24.48	25.59	5749357.0	145200289.0	2009	3	9
1	10/03/09	25.68	26.95	25.68	26.87	7507770.0	198480965.0	2009	3	10
2	11/03/09	26.50	26.95	26.26	26.64	5855095.0	155815439.0	2009	3	11
3	12/03/09	26.15	26.47	25.82	26.18	6294955.0	164489409.0	2009	3	12
4	13/03/09	26.01	26.24	25.65	25.73	6814568.0	176228331.0	2009	3	13

```
[9]: df.head()
```

[9]:

	Date	Open	High	Low	Close	Traded Volume	Turnover	year	Month	Day
0	09/03/09	25.16	25.82	24.48	25.59	5749357.0	145200289.0	2009	3	9
1	10/03/09	25.68	26.95	25.68	26.87	7507770.0	198480965.0	2009	3	10
2	11/03/09	26.50	26.95	26.26	26.64	5855095.0	155815439.0	2009	3	11
3	12/03/09	26.15	26.47	25.82	26.18	6294955.0	164489409.0	2009	3	12
4	13/03/09	26.01	26.24	25.65	25.73	6814568.0	176228331.0	2009	3	13

```
[10]: # predict whether the stock price will go up or down
df["Target"] = (df['Close'].shift(-1) > df['Close']).astype(int)  # 1 if price goes up, 0 if down
df.head()
```

[10]:

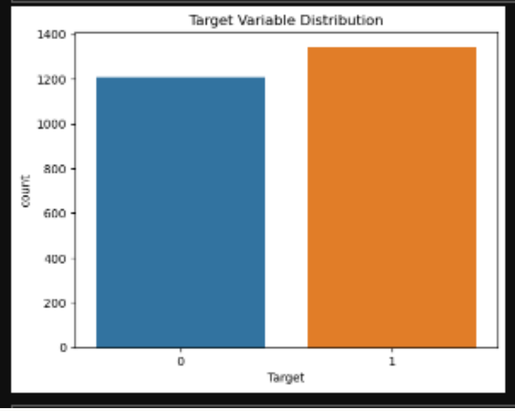
	Date	Open	High	Low	Close	Traded Volume	Turnover	year	Month	Day	Target
0	09/03/09	25.16	25.82	24.48	25.59	5749357.0	145200289.0	2009	3	9	1
1	10/03/09	25.68	26.95	25.68	26.87	7507770.0	198480965.0	2009	3	10	0
2	11/03/09	26.50	26.95	26.26	26.64	5855095.0	155815439.0	2009	3	11	0
3	12/03/09	26.15	26.47	25.82	26.18	6294955.0	164489409.0	2009	3	12	0
4	13/03/09	26.01	26.24	25.65	25.73	6814568.0	176228331.0	2009	3	13	1

```
[11]: # finding the dataset is balanced or not
df["Target"].value_counts()
```

```
[11]: Target
1    1343
0    1287
Name: count, dtype: int64
```

```
[12]: #visualising the target feature
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x=df['Target'])
plt.title("Target Variable Distribution")
plt.show()
```



Code

[13]:
```python
#preprocessing libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

#importing model and evaluation libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,confusion_matrix

# Define features (excluding 'Date' and the target variable)
X = df.drop(columns=['Date', 'Target'])
y = df['Target']  # Target variable (0 or 1)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature Scaling (important for logistic regression)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Model evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 0.5313725490196078
              precision    recall  f1-score   support

           0       0.51      0.19      0.28       248
           1       0.54      0.83      0.65       270

    accuracy                           0.53       518
   macro avg       0.52      0.51      0.47       518
weighted avg       0.52      0.53      0.48       518

Confusion Matrix:
 [[ 46 194]
 [ 45 225]]
```

[15]:
```python
# Import necessary libraries
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,confusion_matrix
from sklearn.feature_selection import SelectKBest, f_classif

# Define features (excluding 'Date' and the target variable)
X = df.drop(columns=['Date', 'Target'])
y = df['Target']  # Target variable (0 or 1)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature Selection using ANOVA F-score (f_classif)
selector = SelectKBest(score_func=f_classif, k=5)  # Select top 5 features
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

# Check selected features
selected_features = X.columns[selector.get_support()]
print("Selected Features:", selected_features)

# Feature Scaling (important for logistic regression)
scaler = StandardScaler()
X_train_selected = scaler.fit_transform(X_train_selected)
X_test_selected = scaler.transform(X_test_selected)

# Define Logistic Regression model
model = LogisticRegression()

# Define hyperparameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],  # Regularization strength
    'penalty': ['l1', 'l2', 'elasticnet', None],  # Regularization type
    'solver': ['liblinear', 'saga']  # Solvers that support L1 and L2 penalties
}

# Perform Grid Search with cross-validation
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train_selected, y_train)

# Best model from grid search
best_model = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)

# Predictions using the best model
y_pred = best_model.predict(X_test_selected)

# Model evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

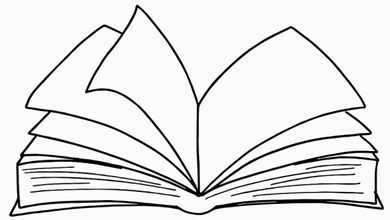
```
Selected Features: Index(['Open', 'Traded Volume', 'year', 'Month', 'Day'], dtype='object')
Best Parameters: {'C': 0.01, 'penalty': 'l1', 'solver': 'saga'}
Accuracy: 0.5294117647058824
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       248
           1       0.53      1.00      0.69       270

    accuracy                           0.53       518
   macro avg       0.26      0.50      0.35       518
weighted avg       0.28      0.53      0.37       518

Confusion Matrix:
 [[  0 248]
 [  0 270]]
```

References



Raja Lakshmi Eduverse Data Science Course – Mrs. Gowthami Shyam, Instructor

- Course: Data Science
- Institution: Rajalakshmi Eduverse
- This course provided foundational knowledge and practical skills in data science, including data preprocessing, machine learning algorithms, and model evaluation.

Kaggle (2025). "Stock Price Dataset."

Retrieved from https://www.kaggle.com/datasets

The dataset used for this project was sourced from Kaggle, which provided relevant stock price data for analysis and model development.


Thank You