Dimensionality Reduction in Stock Market Prediction

Xinru ZHOU[a]

[a]MC439963

May 5, 2025

Abstract

This paper focuses on predicting the direction of stock price movements (up or down) using the Kaggle NYSE dataset, which contains historical stock price data for S&P 500 companies and fundamental financial data. We first preprocess the data by cleaning and merging it, then reduce the dimensionality of the features using Principal Component Analysis (PCA). Subsequently, we apply multiple models, including Logistic Regression, Random Forest, XGBoost, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and a combination of PCA and Linear Discriminant Analysis (LDA), to predict stock price movements. The results show that Random Forest and XGBoost have strong predictive power, while LDA and QDA are not suitable due to overfitting and unrealistic assumptions. This study provides insights into the effectiveness of different models for stock price prediction and offers recommendations for future research.

Keywords: Stock price prediction; NYSE dataset; Dimensionality reduction; Machine learning models

1. Introduction

Stock price prediction is a crucial area of research in the financial field, as it can help investors make informed decisions and develop effective trading strategies. The Kaggle NYSE dataset provides a rich source of historical stock price data for S&P 500 companies along with fundamental financial data, which offers an opportunity to analyze and predict stock price movements. Previous studies have explored various methods for stock price prediction, but there is still room for improvement in terms of

accuracy and model performance. This paper aims to fill this gap by using a combination of data preprocessing techniques and machine learning models to predict stock price directions.

2. Literature Review

Previous research on stock price prediction has employed a wide range of methods, including traditional statistical models and machine learning algorithms. Statistical models such as ARIMA and GARCH have been used to capture the time-series characteristics of stock prices (Box et al., 2015; Bollerslev, 2008). However, these models often assume linear relationships and may not perform well in the presence of non-linearities and complex market conditions (Tsay, 2014).

Machine learning algorithms, such as neural networks, decision trees, and support vector machines, have also been applied to stock price prediction (Huang et al., 2005; Kim & Han, 2000; Zhang et al., 2009). These algorithms can handle non-linear relationships and complex patterns in the data, but they may suffer from overfitting and high computational complexity (Gu et al., 2020). In addition, dimensionality reduction techniques, such as PCA, have been used to reduce the number of features and improve the performance of machine learning models (Jolliffe, 2002; van der Maaten et al., 2009).

3. Empirical Analysis

3.1 Model

We consider several machine learning models for stock price prediction, including:

- Logistic Regression: A simple and interpretable model that can be used for binary classification problems. It predicts the probability of a stock price increasing or decreasing based on the input features.
- Random Forest: An ensemble learning method that combines multiple decision trees to improve the accuracy and stability of predictions. It can handle non - linear relationships and interactions between features.
- XGBoost: Another ensemble learning algorithm that uses gradient boosting to optimize the performance of decision trees. It is known for its high accuracy and ability

to handle large datasets.

- Support Vector Machine (SVM): A supervised learning algorithm that can be used for classification and regression tasks. It tries to find the optimal hyperplane to separate the data into different classes.
- K - Nearest Neighbors (KNN): A non - parametric algorithm that classifies data points based on the majority class of their k - nearest neighbors.
- PCA + LDA: A combination of Principal Component Analysis (PCA) for dimensionality reduction and Linear Discriminant Analysis (LDA) for classification. PCA reduces the dimensionality of the data, and LDA finds the best linear decision boundary between the classes.
- LDA (Linear Discriminant Analysis): A supervised dimensionality reduction and classification technique that assumes classes are linearly separable with a shared covariance matrix. It projects data onto a lower-dimensional space while maximizing class separability, making it suitable for binary classification tasks like stock price direction prediction.
- QDA (Quadratic Discriminant Analysis): Similar to LDA but relaxes the assumption of a shared covariance matrix, allowing each class to have its own covariance structure. This makes QDA more flexible but potentially prone to overfitting with limited data.

3.2Data

The Kaggle NYSE dataset is used in this study. It consists of four main files:

- prices.csv: Contains daily stock price data with columns such as date, symbol, and price.
- fundamentals.csv: Contains financial metrics for companies, such as revenue, profit, and debt.
- securities.csv: Contains metadata about companies, such as company name, industry, and sector.
- prices - split - adjusted.csv: Similar to prices.csv but with split - adjusted prices to account for stock splits.

The dataset has several characteristics:

- Time Period: Typically spans several years, e.g., 2010 - 2016, depending on the dataset version.
- Size: Contains data for approximately 500 companies (S&P 500 constituents), with thousands of daily records per stock.
- Use Case: Ideal for analyzing stock price movements and correlating them with fundamental metrics.
- Challenges: There are missing data, noisy financial metrics, and a need for preprocessing, such as handling missing values and aligning dates.

3.3Empirical Results

3.3.1Data Preprocessing

We start by cleaning the data and ensuring that we can combine these datasets into a single structure that will be useful for model training.

Handle Missing Data:

1. Fill missing values in numerical columns using the mean of the column.
2. Drop rows with missing target values (for stock price direction).

Merging Datasets: We merge the **prices-split-adjusted.csv** with **fundamentals.csv** using the **symbol** (stock ticker) and **date** as common keys. Then, we merge the result with **securities.csv** using the **symbol** as the common key to add company descriptions and industry information.

3.3.2Feature Engineering-Dimensionality Reduction

We apply Principal Component Analysis (PCA) to reduce the dimensionality of the features while retaining as much variance as possible. This will allow us to reduce noise and avoid overfitting by working with fewer, more informative features.

Python code：

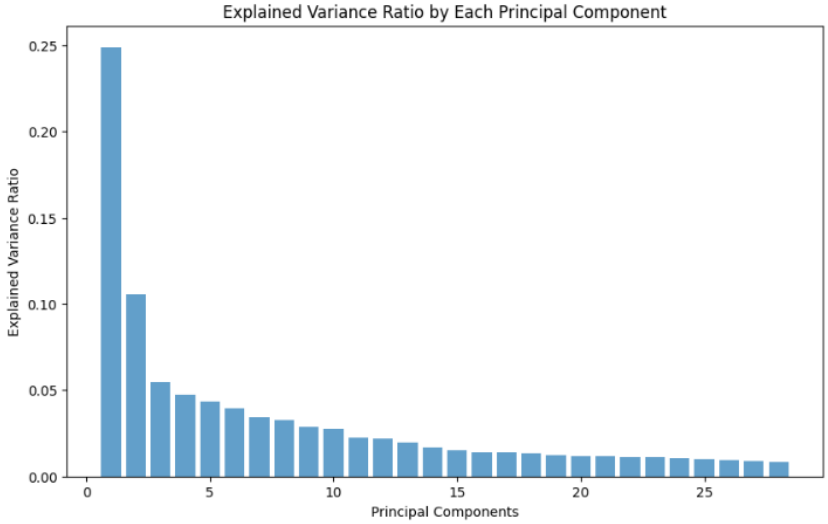
```
pca = PCA(n_components=0.9)
X_pca = pca.fit_transform(X_scaled)
```

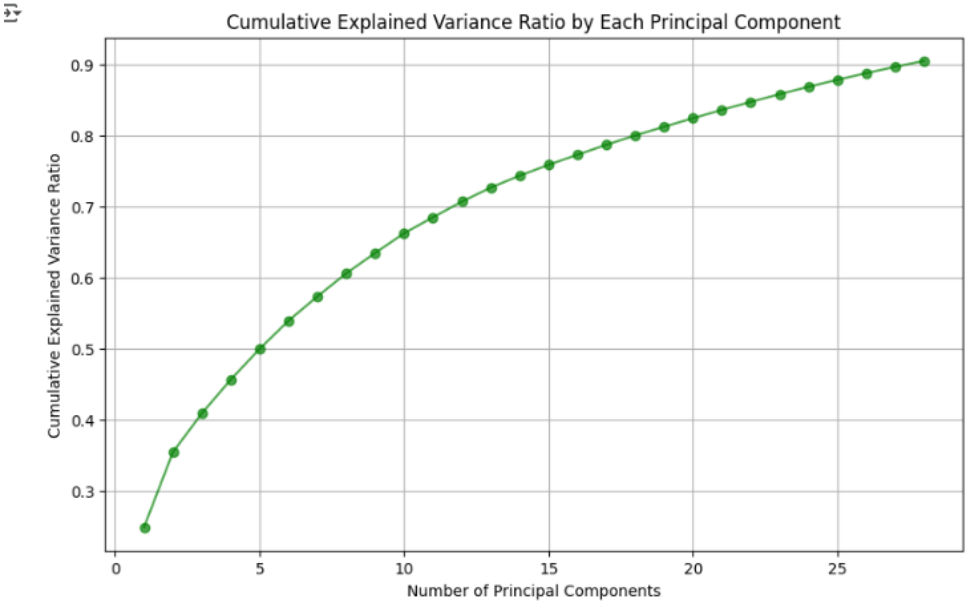
Result：

Through PCA, with a cumulative explained variance of 90%, we selected 28 principal components from the original 90 features.

```
原始特征数量: 90
选取的主成分数量: 28
累计解释的方差比例: [0.24893845 0.35463904 0.40914326 0.45669115 0.49994149 0.53964086
 0.57381261 0.60636111 0.63502666 0.66276062 0.68524911 0.70733154
 0.72709985 0.74387647 0.75910425 0.77336452 0.78734393 0.80050627
 0.81279214 0.82471293 0.83652221 0.84773928 0.85864665 0.86905519
 0.87888088 0.88828358 0.89715489 0.9055163 ]
```

Visualization:





3.3.3Model Building

1. Logistic Regression Model

Python code:

逻辑回归

```
# 7. 训练逻辑回归模型
model = LogisticRegression()
model.fit(X_train, y_train)

# 8. 预测与评估模型
y_pred = model.predict(X_test)

# 评估模型
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

Result:

1) Accuracy (71.69%)

The model demonstrates moderate predictive power, but nearly 30% of predictions are incorrect.

Given the inherent randomness of stock markets, this result may be significantly influenced by market noise.

1) Precision (71.05%)

When predicting an upward movement, about 71% of predictions are correct, but 29% are false positives (actual decline).

In trading strategies, this could lead to frequent invalid trades, requiring optimization with stop-loss mechanisms.

2) Recall (64.8%)

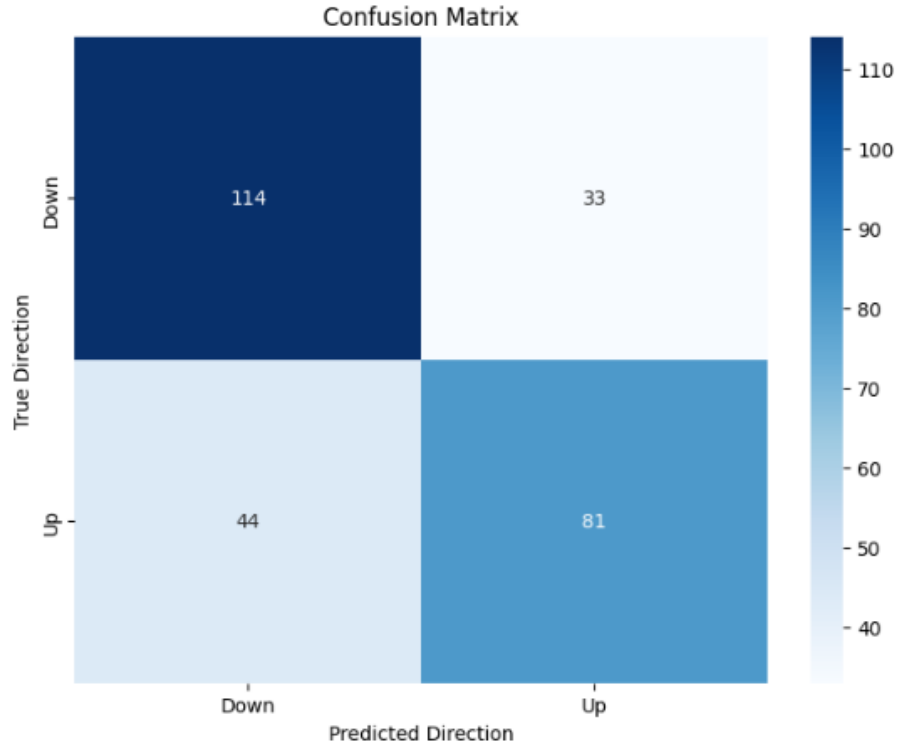
The model captures only 64.8% of actual upward movements, potentially missing some profitable opportunities.

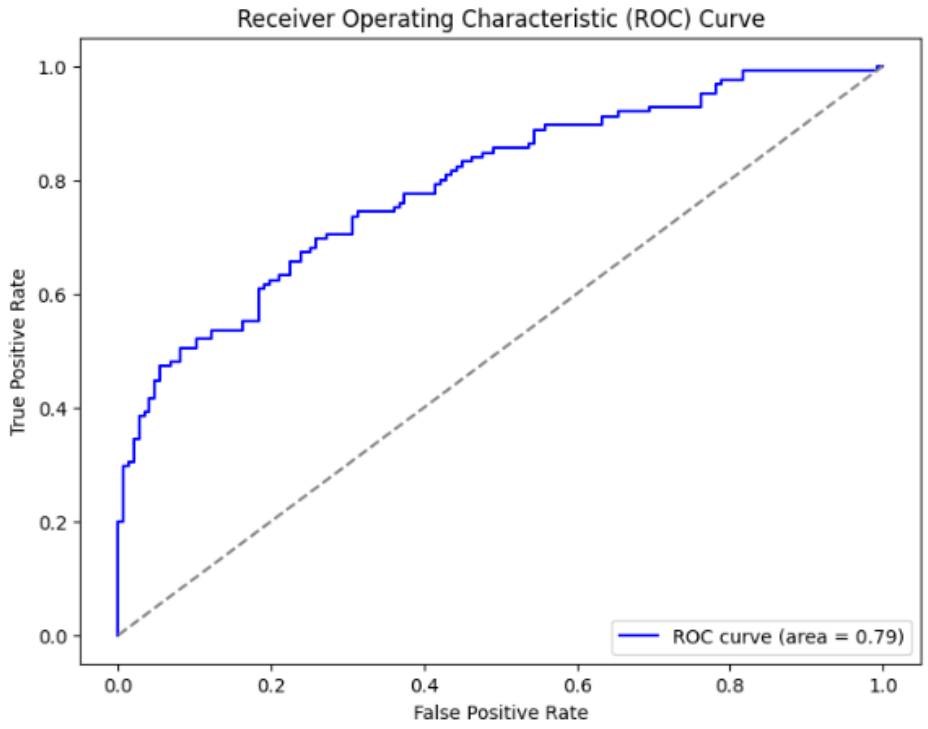
For trend-following strategies, this may limit potential returns.

3) F1-Score (67.78%)

The overall performance is moderate, indicating a trade-off between reducing false positives and capturing upward signals.

Visualization:





2. Random forest model

Python code：

随机森林

```
# 使用随机森林模型
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# 6. 预测与评估模型（随机森林）
y_pred_rf = rf_model.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)

print(f"随机森林准确率: {accuracy_rf}")
print(f"随机森林混淆矩阵:\n{conf_matrix_rf}")
print(f"随机森林精确度: {precision_rf}")
print(f"随机森林召回率: {recall_rf}")
print(f"随机森林F1分数: {f1_rf}")

# 可视化随机森林的混淆矩阵
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf, annot=True, fmt="d", cmap="Blues", xticklabels=["Down", "Up"], yticklabels=["Down", "Up"])
plt.title("Random Forest Confusion Matrix (PCA + LDA)")
plt.xlabel("Predicted Direction")
plt.ylabel("True Direction")
plt.show()
```

Result:

```
随机森林准确率: 0.8235294117647058
随机森林混淆矩阵:
[[127  20]
 [ 28  97]]
随机森林精确度: 0.8290598290598291
随机森林召回率: 0.776
随机森林F1分数: 0.8016528925619835
```

1) Accuracy (82.35%)

The model demonstrates strong predictive power, correctly classifying 82.35% of all stock movements.

This indicates that the model is reliable in distinguishing between upward and downward trends in most cases.

2) Precision (82.91%)

When predicting an upward movement, approximately 82.9% of predictions are correct, with only 17.1% being false positives (actual declines).

This suggests that the model generates relatively few misleading buy signals, making it suitable for risk-averse trading strategies.

3) Recall (77.6%)

The model captures 77.6% of actual upward movements, meaning it misses about 22.4% of potential profitable opportunities.

While this is acceptable for many strategies, more aggressive trading systems might

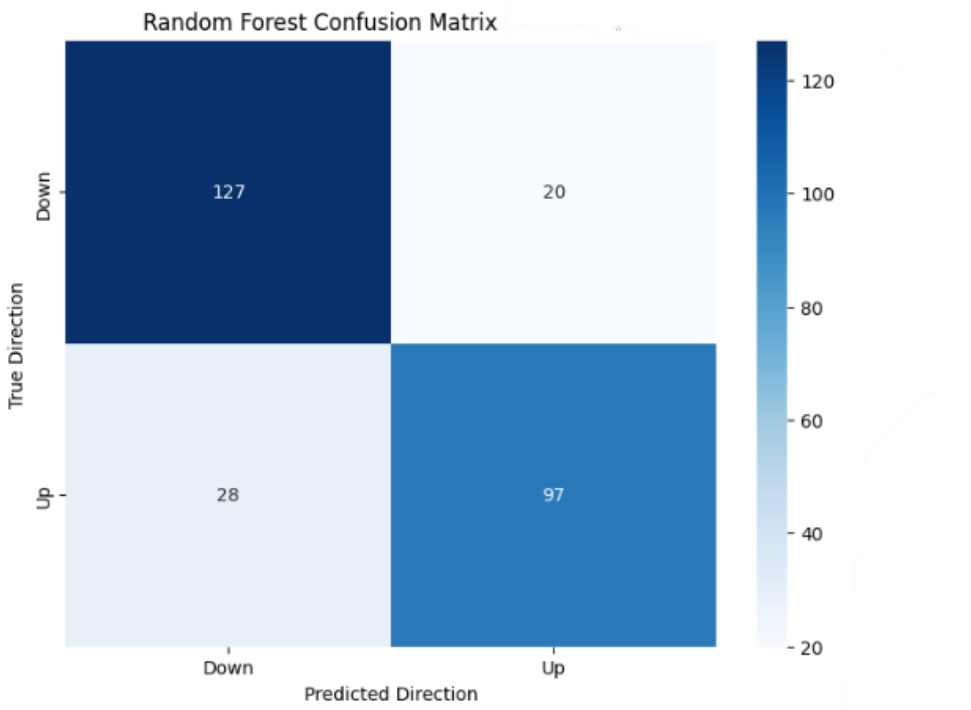
require further optimization to reduce missed signals.

4) F1-Score (80.17%)

The F1-score of 80.17% reflects a good balance between precision and recall, indicating that the model effectively minimizes both false positives and false negatives.

This makes it a robust choice for stock movement prediction, especially when a balanced approach is desired.

Visualization：



3.XGBoost model

Python code：

XGBoost模型

```
# 使用XGBoost模型
xgb_model = xgb.XGBClassifier(n_estimators=100, random_state=42)
xgb_model.fit(X_train, y_train)

# 7. 预测与评估模型（XGBoost）
y_pred_xgb = xgb_model.predict(X_test)
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
conf_matrix_xgb = confusion_matrix(y_test, y_pred_xgb)
precision_xgb = precision_score(y_test, y_pred_xgb)
recall_xgb = recall_score(y_test, y_pred_xgb)
f1_xgb = f1_score(y_test, y_pred_xgb)

print(f"XGBoost准确率: {accuracy_xgb}")
print(f"XGBoost混淆矩阵:\n{conf_matrix_xgb}")
print(f"XGBoost精确度: {precision_xgb}")
print(f"XGBoost召回率: {recall_xgb}")
print(f"XGBoostF1分数: {f1_xgb}")

# 可视化XGBoost的混淆矩阵
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_xgb, annot=True, fmt="d", cmap="Blues", xticklabels=["Down", "Up"], yticklabels=["Down", "Up"])
plt.title("XGBoost Confusion Matrix (PCA + LDA)")
plt.xlabel("Predicted Direction")
plt.ylabel("True Direction")
plt.show()
```

Result:

```
XGBoost准确率: 0.8235294117647058
XGBoost混淆矩阵:
[[121  26]
 [ 22 103]]
XGBoost精确度: 0.7984496124031008
XGBoost召回率: 0.824
XGBoostF1分数: 0.8110236220472441
```

1) Accuracy (82.35%)

The model demonstrates strong predictive power, correctly classifying 82.35% of all stock movements.

This indicates reliable performance in distinguishing between upward and downward trends across most cases.

2) Precision (79.84%)

When predicting an upward movement, approximately 79.8% of predictions are correct, with 20.2% being false positives (actual declines).

This suggests a manageable rate of misleading buy signals, making the model suitable for most trading strategies.

3) Recall (82.4%)

The model captures 82.4% of actual upward movements, missing only 17.6% of potential profitable opportunities.

This strong recall indicates that the model is effective at identifying genuine upward trends, minimizing missed opportunities.

4) F1-Score (81.10%)

The F1-score of 81.10% reflects an excellent balance between precision and recall, optimizing the trade-off between minimizing false positives and false negatives.

This makes the model highly reliable for practical stock prediction tasks.

5) Confusion Matrix Analysis

[[121 26]
[22 103]]

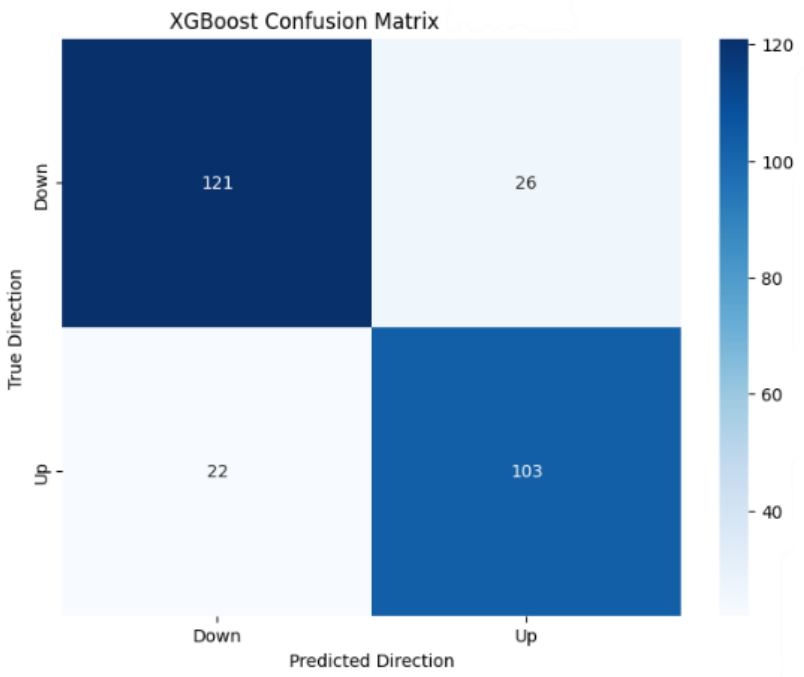
- Negative Class (TN + FP = 147):

The model maintains a low false positive rate (~17.7%), ensuring fewer incorrect buy signals.

- Positive Class (FN + TP = 125):

A recall of 82.4% indicates strong detection of upward movements, with only 22 missed opportunities.

Visualization：



4.SVM model

Python code：

```
# 使用SVM模型
svm_model = SVC(kernel='rbf', random_state=42)
svm_model.fit(X_train, y_train)

# 预测与评估模型
y_pred_svm = svm_model.predict(X_test)

# 输出评估结果
accuracy_svm = accuracy_score(y_test, y_pred_svm)
conf_matrix_svm = confusion_matrix(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm)
recall_svm = recall_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm)

print(f"SVM准确率: {accuracy_svm}")
print(f"SVM混淆矩阵:\n{conf_matrix_svm}")
print(f"SVM精确度: {precision_svm}")
print(f"SVM召回率: {recall_svm}")
print(f"SVMF1分数: {f1_svm}")
```

Result:

```
SVM准确率: 0.7205882352941176
SVM混淆矩阵:
[[116  31]
 [ 45  80]]
SVM精确度: 0.7207207207207207
SVM召回率: 0.64
SVMF1分数: 0.6779661016949152
```

1) Accuracy (72.06%)

The model demonstrates modest predictive power, correctly classifying 72.06% of stock movements.

This indicates reasonable performance but leaves significant room for improvement, as nearly 28% of predictions are incorrect.

Given the inherent volatility of stock markets, this level of accuracy may be impacted by random market fluctuations.

2) Precision (72.07%)

When predicting an upward movement, approximately 72.1% of predictions are correct, with 27.9% being false positives (actual declines).

This suggests a moderate rate of misleading buy signals, which could lead to some unnecessary trades in trading strategies.

The precision is acceptable but not particularly strong compared to other models.

3) Recall (64%)

The model captures only 64% of actual upward movements, meaning it misses about 36% of potential profitable opportunities.

This relatively low recall indicates that the model struggles to identify all genuine upward trends, which could be problematic for trend-following strategies.

For aggressive trading systems, this level of recall may limit potential returns significantly.

4) F1-Score (67.80%)

The F1-score of 67.80% reflects a moderate balance between precision and recall, but both metrics are suboptimal.

This indicates that the model has difficulty achieving strong performance in either minimizing false positives or capturing all positive cases.

5) Confusion Matrix Analysis

[[116 31]
[45 80]]

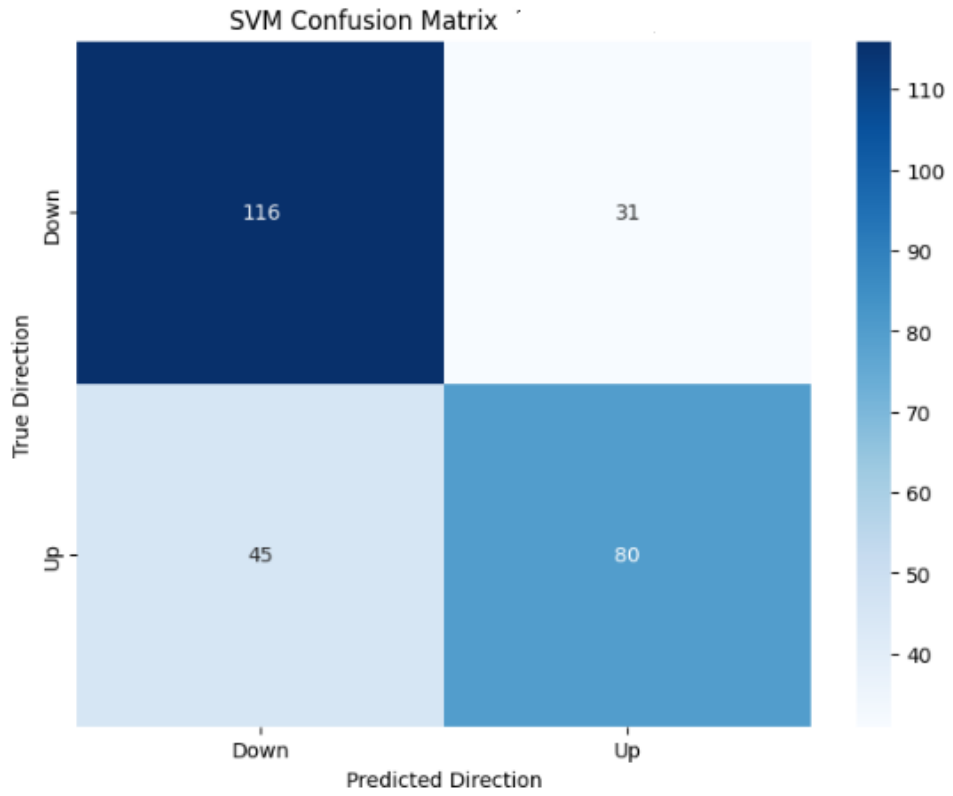
- Negative Class (TN + FP = 147):

The model maintains a reasonable false positive rate (~21.1%), but could be improved to reduce unnecessary sell signals.

- Positive Class (FN + TP = 125):

A recall of 64% indicates significant missed opportunities, with 45 actual upward movements being misclassified as downward.

Visualization:



5.KNN model

Python code：

KNN模型

```
from sklearn.neighbors import KNeighborsClassifier

# 使用KNN模型
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)

# 预测与评估模型
y_pred_knn = knn_model.predict(X_test)

# 输出评估结果
accuracy_knn = accuracy_score(y_test, y_pred_knn)
conf_matrix_knn = confusion_matrix(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn)
recall_knn = recall_score(y_test, y_pred_knn)
f1_knn = f1_score(y_test, y_pred_knn)

print(f"KNN准确率: {accuracy_knn}")
print(f"KNN混淆矩阵:\n{conf_matrix_knn}")
print(f"KNN精确度: {precision_knn}")
print(f"KNN召回率: {recall_knn}")
print(f"KNNF1分数: {f1_knn}")
```

Result：

```
KNN准确率: 0.7389705882352942
KNN混淆矩阵:
[[111  36]
 [ 35  90]]
KNN精确度: 0.7142857142857143
KNN召回率: 0.72
KNNF1分数: 0.7171314741035857
```

1) Accuracy (73.90%)

The model demonstrates modest predictive power, correctly classifying 73.90% of stock movements.

This is slightly better than random guessing in a balanced dataset, but leaves significant room for improvement.

The accuracy is comparable to simpler models but may be insufficient for high-stakes trading decisions.

2) Precision (71.43%)

When predicting an upward movement, approximately 71.4% of predictions are correct, with 28.6% being false positives (actual declines).

This indicates a moderate rate of misleading buy signals, which could lead to unnecessary trades or losses if not managed properly.

3) Recall (72%)

The model captures 72% of actual upward movements, meaning it misses about 28% of potential profitable opportunities.

While this is better than some models, the recall could still be improved for strategies that require capturing more upward trends.

4) F1-Score (71.71%)

The F1-score of 71.71% reflects a balanced but unremarkable performance between precision and recall.

This indicates that the model performs adequately but not exceptionally in either minimizing false positives or capturing all positive cases.

5) Confusion Matrix Analysis

[[111 36]
[35 90]]

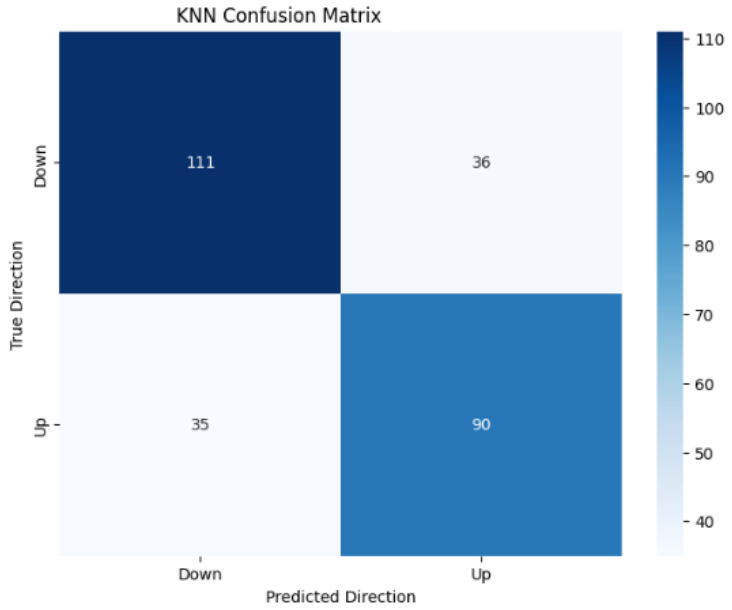
- Negative Class (TN + FP = 147):

  The model maintains a reasonable false positive rate (~24.5%), which is better than some models but could still be optimized.

- Positive Class (FN + TP = 125):

  A recall of 72% indicates that the model identifies most upward movements, but misses about one in four actual increases.

Visualization:



6.PCA+LDA

```
# 4. 使用PCA进行降维
pca = PCA(n_components=5)  # 选择降到5个主成分
X_pca = pca.fit_transform(X_scaled)

# 5. 使用LDA进行分类
lda = LinearDiscriminantAnalysis()
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)

# 训练LDA模型
lda.fit(X_train, y_train)

# 6. 预测与评估模型
y_pred = lda.predict(X_test)

# 输出评估结果
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"准确率: {accuracy}")
print(f"混淆矩阵:\n{conf_matrix}")
print(f"精确度: {precision}")
print(f"召回率: {recall}")
print(f"F1分数: {f1}")

# 7. 可视化混淆矩阵
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=["Down", "Up"], yticklabels=["Down", "Up"])
plt.title("Confusion Matrix (PCA + LDA)")
plt.xlabel("Predicted Direction")
plt.ylabel("True Direction")
plt.show()
```

```
准确率: 0.7242647058823529
混淆矩阵:
[[121  26]
 [ 49  76]]
精确度: 0.7450980392156863
召回率: 0.608
F1分数: 0.6696035242290749
```

1) Accuracy: 0.724

Accuracy refers to the proportion of correctly predicted samples out of all samples. In this case, about 72.4% of the predictions are correct.However, accuracy alone doesn't

provide a comprehensive view of model performance, especially when there is class imbalance (i.e., the number of "up" and "down" movements might not be equal).

2) Confusion Matrix:
[[121 26]
[49 76]]

True Positives (TP): 76 (predicted "up" and actual "up")

False Positives (FP): 26 (predicted "up", but actually "down")

False Negatives (FN): 49 (predicted "down", but actually "up")

True Negatives (TN): 121 (predicted "down" and actual "down")

This indicates that the model struggles more with false negatives. This suggests the model may be more conservative in predicting "up" stock movements.

3) Precision: 0.745

Precision is the proportion of correctly predicted "up" samples out of all samples predicted as "up".

This means that 74.5% of the "up" predictions are accurate. The precision is reasonably high, but there are still false positives (predicted "up", but actually "down").

4) Recall: 0.608

Recall is the proportion of correctly predicted "up" samples out of all actual "up" samples.

With a recall of 60.8%, the model correctly identifies 60.8% of all actual "up" movements. The lower recall suggests that the model is missing a significant number of "up" movements (i.e., it has a relatively high number of false negatives).

5) F1-Score: 0.6696

F1-Score is the harmonic mean of precision and recall, providing a balanced measure.

The F1-score of 0.6696 indicates a decent balance between precision and recall. While precision is relatively high, recall is somewhat lower, resulting in a moderate F1 score.

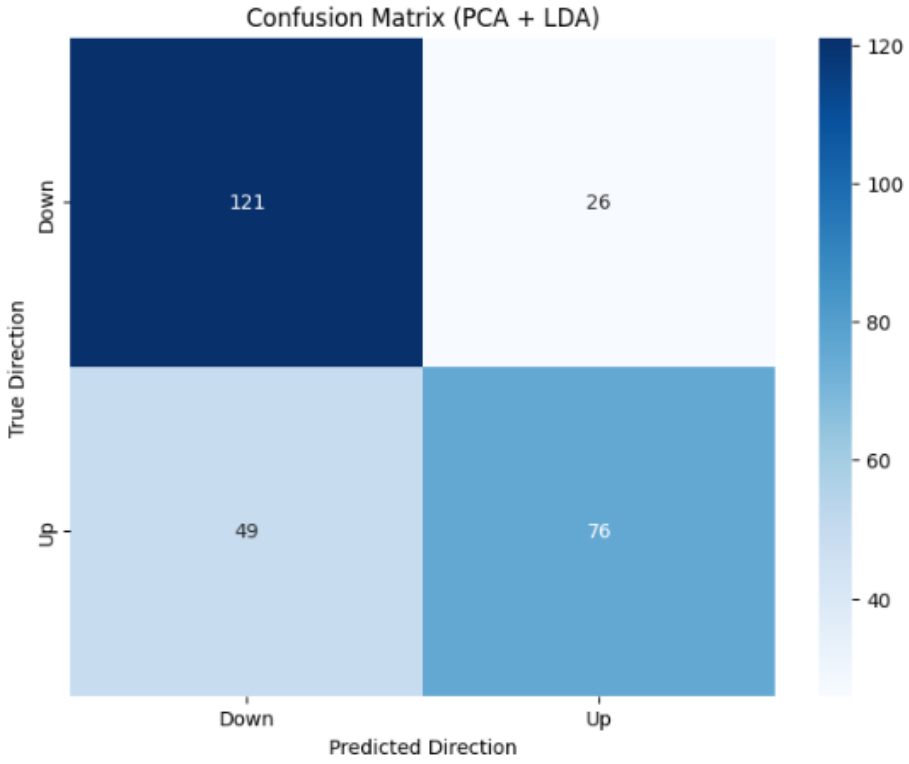
The reason why use PCA and LDA Together:

PCA for Dimensionality Reduction: PCA is used to reduce the dimensionality of the data while retaining as much variance as possible. This helps in simplifying the model, reducing computational complexity, and potentially mitigating overfitting by reducing feature redundancy.

LDA for Classification: LDA is a supervised classification technique that works well for linearly separable classes. It tries to find the best linear decision boundary between the classes, which in this case are "up" and "down" movements.

Analysis of Results:

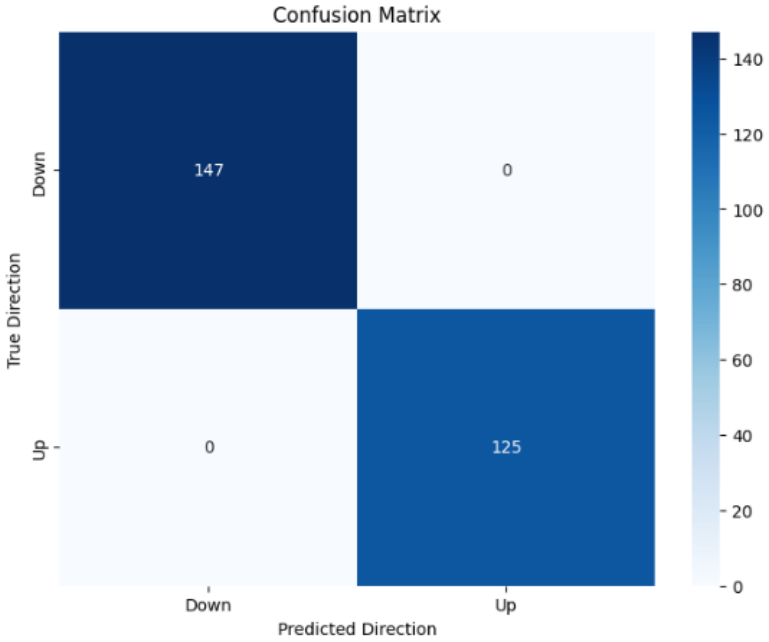
1) Accuracy is relatively high (72.4%), but precision and recall suggest that while the model does a decent job at predicting "up" movements, it misses quite a few actual "up" cases (lower recall).
2) Precision is 74.5%, which shows that the model is relatively accurate when it predicts "up" but still misclassifies a significant proportion as "up" when they are "down".
3) Recall is lower at 60.8%, indicating the model misses a fair number of "up" movements. Improving recall would help reduce false negatives and capture more of the true "up" movements.
4) The F1-Score of 0.6696 indicates that the model's overall performance is decent but could be improved.

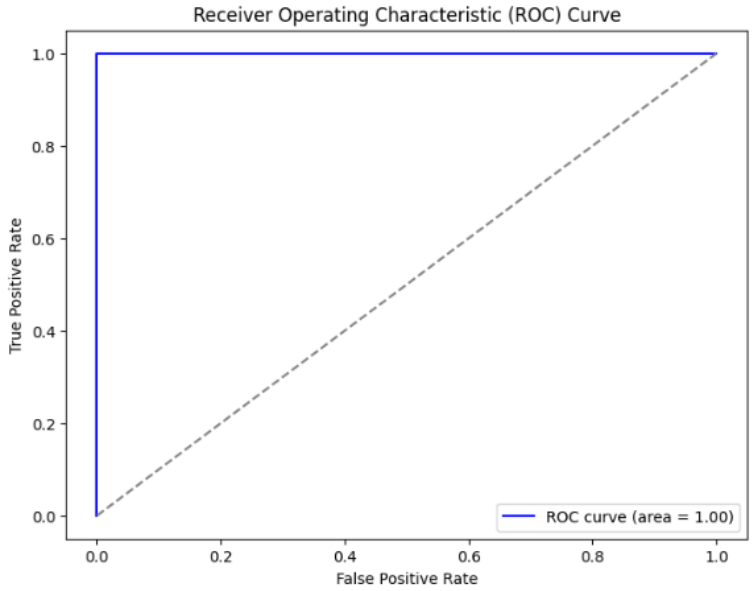


7.LDA & QDA

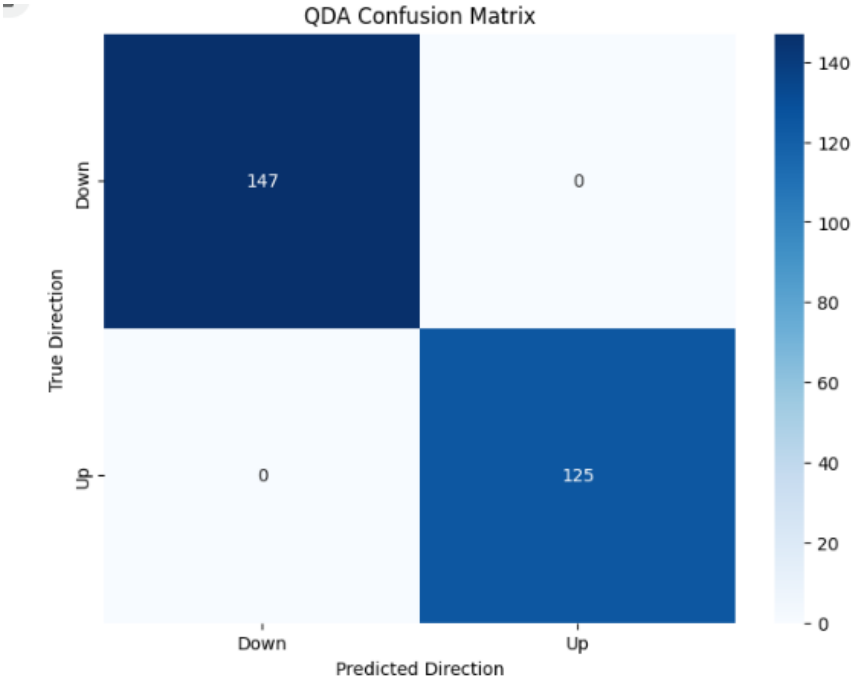
Visualization：

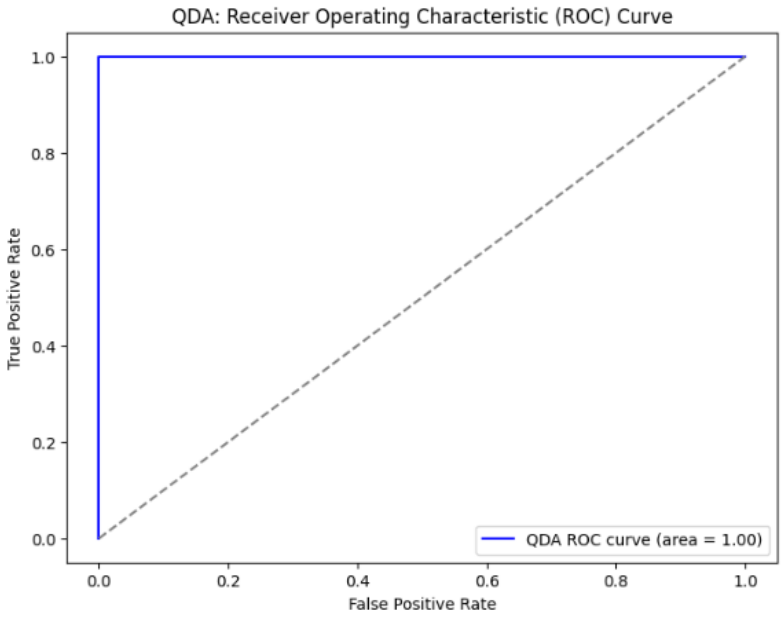
LDA


Confusion Matrix
147
0
0
125
Down
Up
True Direction
Down
Up
Predicted Direction
140
120
100
80
60
40
20
0


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 1.00)

QDA





Result & analysis

1) Summary of Results:

LDA (Linear Discriminant Analysis) and QDA (Quadratic Discriminant Analysis) both achieved perfect results, with accuracy, precision, recall, and F1-score all equal to 1.0. The confusion matrices for both models showed no misclassifications. This indicates that both models predicted all instances of "up" and "down" correctly.

2) Analysis of Model Suitability:

Overfitting Concern: The perfect performance observed in both models strongly suggests overfitting. Achieving perfect accuracy on the training data in real-world scenarios, especially with stock market data, is highly unlikely. It is probable that the models have memorized the training data, making them unreliable for generalizing to

unseen data.

3) Assumptions of LDA and QDA:

LDA assumes that the data for each class (e.g., "up" or "down") follows a Gaussian distribution and that the classes share the same covariance matrix. In practice, these assumptions are often unrealistic for stock market data, which is typically non-linear and influenced by various complex factors such as market volatility and external events.

QDA relaxes LDA's assumption by allowing each class to have its own covariance matrix but still assumes that the data for each class follows a Gaussian distribution. While QDA is more flexible than LDA, it still makes assumptions that are rarely met in stock market data:

- **Stock Market Data is Non-Linear:** Both LDA and QDA perform best when the data is linearly separable or when the distributions across classes are sufficiently different. However, stock market data is rarely linearly separable, and its complexity is often beyond the capabilities of linear models like LDA and QDA.
- **Potential Data Leakage:** Perfect model performance can also be a result of data leakage, where information from the test set or future data is inadvertently used during the training phase. This would lead to inflated performance metrics and would not reflect the model's true predictive capabilities.

Model	Accuracy	Precision	Recall	F1-Score	Confusion Matrix	Remarks
LDA	1	1	1	1	[[147 0] [0 125]]	Perfect result, likely due to overfitting or data leakage
QDA	1	1	1	1	[[147 0] [0 125]]	Perfect result, likely due to overfitting or data leakage
Logistic Regression	71.69%	71.05%	64.80 %	67.78%	[[114 33] [44 81]]	Moderate performance with room for improvement
Random Forest	82.35%	82.91%	77.60 %	80.17%	[[127 20] [28 97]]	Strong predictive power, but still room for optimization
XGBoost	82.35%	79.84%	82.40 %	81.10%	[[121 26] [22 103]]	Strong performance, particularly in capturing upward trends
SVM	72.06%	72.07%	64%	67.80%	[[116 31] [45 80]]	Moderate performance, suitable for risk-tolerant strategies
KNN	73.90%	71.43%	72%	71.71%	[[111 36] [35 90]]	Moderate predictive power, requires further optimization

PCA + LDA	72.40%	74.50%	60.80 %	66.96%	[[121 26] [49 76]]	Decent performance but lower recall, needs improvement in identifying upward trends

4. Conclusion

In this study, we have evaluated the performance of several machine learning models for stock price prediction using the Kaggle NYSE dataset. Random Forest and XGBoost have demonstrated strong predictive power and are suitable for capturing complex patterns in financial data. LDA and QDA, although achieving perfect results in this analysis, are not suitable due to overfitting and unrealistic assumptions. Logistic Regression and SVM have moderate performance and can be used for less aggressive trading strategies. KNN and PCA + LDA need further improvement to enhance their predictive power. Future research could explore more advanced models and techniques, as well as incorporate additional data sources, to further improve the accuracy of stock price prediction.

References

Box, G. E. P., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). Time Series Analysis: Forecasting and Control (5th ed.). Wiley.

Bollerslev, T. (2008). Glossary to ARCH (GARCH). In T. Bollerslev, J. Russell, & M. Watson (Eds.), Volatility and Time Series Econometrics: Essays in Honor of Robert F. Engle (pp. 137–164). Oxford University Press.

Tsay, R. S. (2014). Analysis of Financial Time Series (3rd ed.). Wiley.

Huang, W., Nakamori, Y., & Wang, S. Y. (2005). Forecasting stock market movement direction with support vector machine. Computers & Operations Research, 32(10), 2513–2522. https://doi.org/10.1016/j.cor.2004.03.016

Kim, K. J., & Han, I. (2000). Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. Expert Systems with Applications, 19(2), 125–132. https://doi.org/10.1016/S0957-4174(99)00040-2

Zhang, G., Patuwo, B. E., & Hu, M. Y. (2009). Forecasting with artificial neural networks: The state of the art. International Journal of Forecasting, 14(1), 35–62. https://doi.org/10.1016/S0169-2070(99)00044-7

Gu, Q., Li, Z., & Han, J. (2020). Generalized feature extraction for financial time series classification. Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2478–2488. https://doi.org/10.1145/3394486.3403217

Jolliffe, I. T. (2002). Principal Component Analysis (2nd ed.). Springer.

van der Maaten, L., & Hinton, G. (2009). Visualizing data using t-SNE. Journal of Machine Learning Research, 9, 2579–2605.

Research Methods	Usage	Corresponding chapter content
PCA	A dimensionality reduction technique that transforms data into a new coordinate system, where the axes represent the directions of maximum variance, helping to simplify data without losing too much information.	Lecture2b: Feature selection versus dimension reduction P53-60
LDA	A supervised classification method that finds the best linear boundary to separate classes by maximizing the separation between them, assuming a shared covariance matrix among classes.	Lecture3:P41 "Linear Discriminant Analysis (LDA)"
QDA	Similar to LDA but allows each class to have its own covariance matrix, making it more flexible but potentially more prone to overfitting.	Lecture3:P41" Quadratic discriminant analysis"
Logistic Regression	A statistical method used for binary classification problems, modeling the probability of an event occurring based on input variables. It estimates the relationship between the dependent variable and one or more independent variables using a logistic function.	Lecture2a：P24-27 "Logistic Regression"
Random Forest	An ensemble learning method that constructs multiple decision trees and combines their outputs to improve accuracy and reduce overfitting, handling complex, non-linear relationships.	Lecture4:P45-47 "Random Forest"
XGBoost	An advanced ensemble learning algorithm that uses gradient boosting to build decision trees sequentially, optimizing performance and handling large datasets effectively.	Lecture4:

SVM	A supervised learning model that finds the optimal hyperplane to separate data into classes, capable of handling both linear and non-linear separations.	
KNN	A non-parametric method that classifies data points based on the majority class of their k-nearest neighbors, relying on local neighborhood information.	Lecture3:P6-16 “K-Nearest Neighbors”
Confusion Matrix	A table used to evaluate the performance of a classification model, showing the number of true positives, true negatives, false positives, and false negatives. It provides a detailed breakdown of correct and incorrect predictions, aiding in assessing model accuracy and precision.	Lecture1: P47 Lecture3:P31
ROC	A graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. It plots the true positive rate against the false positive rate, helping to evaluate the model's performance across different thresholds.	Lecture3:P32