Predicting Candlestick Trends Using Historical Data and Wavelets

Ehsan KhademOlama

June 20, 2024

Abstract

This report presents a method for predicting the current candlestick trend based on historical data using wavelet transformation and a neural network classifier. The prediction focuses on classifying the trend as 'up', 'down', or 'sideways'. The performance of the model is evaluated using accuracy, classification report, and confusion matrix.

1 Introduction

Accurate trend prediction in stock prices is crucial for making informed trading decisions. This report extends previous work on trend identification using wavelets by developing a model to predict the current candlestick trend based on historical data. The approach involves feature extraction, wavelet transformation, and machine learning classification.

2 Methodology

2.1 Data Collection

Historical stock data for Apple Inc. (AAPL) was collected using the yFinance library.

```
import numpy as np
import pandas as pd
import yfinance as yf
import pywt
import matplotlib.pyplot as plt
import mplfinance as mpf
import seaborn as sns
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,
   accuracy_score, confusion_matrix
```

2.2 Wavelet Transformation

The trend is identified using discrete wavelet transform (DWT) with the 'db2' wavelet.

```
class TrendIdentifier:
def __init__(self, wavelet='db2'):
self.wavelet = wavelet

def identify_trend(self, df):

data = df['Close'].values
if len(data) % 2 != 0: # Ensuring even length for DWT
data = np.pad(data, (0, 1), 'constant', constant_values=(
   data[-1],))

cA, cD = pywt.dwt(data, self.wavelet, 'smooth')
approx = pywt.idwt(cA, np.zeros_like(cD), self.wavelet)

# Convert the numpy array 'approx' to a pandas Series
approx_series = pd.Series(approx[:len(df)]) # Ensuring it
   matches the length of df

# Improved trend identification using pandas Series
   methods
kappa = np.where(
(approx_series > approx_series.shift(1)) &
(approx_series > approx_series.shift(2)) &
(approx_series > approx_series.shift(3)),
'up',
np.where(
(approx_series < approx_series.shift(1)) &
(approx_series < approx_series.shift(2)) &
(approx_series < approx_series.shift(3)),
'down',
'sideways'
)
)

return kappa[-1], approx[-1]
```

2.3 Data Fetching

Historical data fetching using yFinance library.

```
def fetch_historical_data(symbol, interval='1d', period='1
   y'):
df = yf.download(symbol, period=period, interval=interval)
df.dropna(inplace=True)
return df
```

2.4 Feature Extraction

New features are generated based on percentage changes and candlestick patterns.

```
def add_percentage_change_features(df, window=3,
    shift_days=2):
df['High_pct_change'] = df['High'].pct_change(periods=
    window).shift(shift_days)
df['Low_pct_change'] = df['Low'].pct_change(periods=window
    ).shift(shift_days)
df['Open_pct_change'] = df['Open'].pct_change(periods=
    window).shift(shift_days)
df['Close_pct_change'] = df['Close'].pct_change(periods=
    window).shift(shift_days)

df['candle_close2open'] = ((df['Close'] - df['Open']) / (
    df['High'] - df['Low'])).shift(shift_days)
df['candle_high2oc'] = ((df['High'] - df[['Open', 'Close'
    ]].max(axis=1)) / (df['High'] - df['Low'])).shift(
    shift_days)
df['candle_high2low2close'] = ((df['High'] - df['Low']) /
    df['Close']).shift(shift_days)
df['candle_low2oc'] = ((df[['Open', 'Close']].min(axis=1)
    - df['Low']) / (df['High'] - df['Low'])).shift(
    shift_days)

df.dropna(inplace=True)
return df
```

2.5 Custom Rolling Function

Manually apply a rolling operation with a DataFrame window.

```
def apply_custom_rolling(df, window_size, func):
results1 = []
results2 = []
# Loop through the DataFrame creating windows and
    applying the custom function
for start in range(len(df) - window_size + 1):
end = start + window_size
window = df.iloc[start:end]
result1, result2 = func(window)
results1.append(result1)
results2.append(result2)

# Append NaNs for the indices that don't complete a
    window
results1 = [None] * (window_size - 1) + results1
results2 = [None] * (window_size - 1) + results2
```

```
return results1, results2

df = fetch_historical_data("AAPL", "1d", "5y")
trend_identifier = TrendIdentifier(wavelet='db24')
rolling_window = 128
df['Trend'], df['approx'] = apply_custom_rolling(df = df,
window_size = rolling_window,
func = trend_identifier.identify_trend)

result_df = add_percentage_change_features(df, window=1,
   shift_days=1)
```

2.6 Model Training

A neural network classifier is trained on the extracted features. The data is split into training and testing sets without shuffling, ensuring that the model is trained on historical data and tested on more recent data to simulate a realistic backtesting scenario.

```
features = result_df[['High_pct_change',
'Low_pct_change',
'Open_pct_change',
'Close_pct_change',
'candle_close2open',
'candle_high2oc',
'candle_low2oc',
'candle_high2low2close']]
labels = result_df['Trend'].map({'up': 1, 'down': -1, '
   sideways': 0})


X_train, X_test, y_train, y_test = train_test_split(
   features, labels,
test_size=0.4, shuffle=False)
# Scale the training and test sets separately
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

mlp_model = MLPClassifier(hidden_layer_sizes=(100, 100),
activation='relu',
solver='sgd',
alpha=1e-5,
learning_rate_init=0.01,
max_iter=200,
warm_start=True)
mlp_model.out_activation_ = 'softmax'
mlp_model.fit(X_train_scaled, y_train)

# Plot the convergence of the neural network
```

```
plt.figure(figsize=(10, 6))
plt.plot(mlp_model.loss_curve_, label='Loss Curve')
plt.title('Neural Network Convergence')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.grid()
plt.show()

y_pred = mlp_model.predict(X_test_scaled)
```

2.7 Model Evaluation

The model's performance is evaluated using a classification report, accuracy score, and confusion matrix.

```
# Print the classification report
print(classification_report(y_test, y_pred, zero_division
   =0))

# Print the accuracy score
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

2.8 Visualization

Visualization of the candlestick chart with predicted trends.

```
result_df['MLP_Trend'] = mlp_model.predict(scaler.
   transform(result_df[['High_pct_change',
'Low_pct_change',
'Open_pct_change',
'Close_pct_change',
'candle_close2open',
'candle_high2oc',
'candle_low2oc',
'candle_high2low2close']].fillna(0)))

crp = 0.01
```

```
arrows_up = np.where(result_df['MLP_Trend'] == 1,
   result_df['High'] * (1+2*crp), np.nan)
arrows_down = np.where(result_df['MLP_Trend'] == -1,
   result_df['Low'] * (1-2*crp), np.nan)

arrows_up_org = np.where(result_df['Trend'] == 'up',
   result_df['High'] * (1+crp), np.nan)
arrows_down_org = np.where(result_df['Trend'] == 'down',
   result_df['Low'] * (1-crp), np.nan)

# Define the additional plots
alphac = 0.45
mkz = 5
apds = [
mpf.make_addplot(result_df['approx'], color='blue', alpha=
    alphac, panel=0,
ylabel='Approximation'),

mpf.make_addplot(arrows_up_org, scatter=True, markersize=
   mkz, marker='^',
color='green', alpha= alphac, panel=0, label='Original␣Up␣
   Trend'),

mpf.make_addplot(arrows_up, scatter=True, markersize=mkz,
   marker='^',
color='black', alpha= alphac, panel=0, label='Predicted␣Up
   ␣Trend'),

mpf.make_addplot(arrows_down_org, scatter=True, markersize
   =mkz, marker='v',
color='red', alpha= alphac, panel=0, label='Original␣Down␣
   Trend'),

mpf.make_addplot(arrows_down, scatter=True, markersize=mkz
   , marker='v',
color='orange', alpha= alphac, panel=0, label='Predicted␣
   Down␣Trend'),
]

mc = mpf.make_marketcolors(up='g', down='r', inherit=True)
s = mpf.make_mpf_style(marketcolors=mc)

mpf.plot(result_df, type='candle',
addplot=apds, style=s,
title='Candlestick␣Chart␣with␣Wavelet␣and␣MLP␣Predicted␣
   Trends',
volume=False,
```

```
            savefig='
               candlestick_chart_with_wavelet_MLP_Predicted_trends.svg
               ')
```

3 Results

3.1 Classification Report

Class	Precision	Recall	F1-Score	Support
-1	0.59	0.58	0.59	161
0	0.30	0.18	0.22	119
1	0.63	0.77	0.69	223
Accuracy			0.57	503
Macro avg	0.51	0.51	0.50	503
Weighted avg	0.54	0.57	0.55	503

Table 1: Classification Report

3.2 Accuracy

Metric	Value
Accuracy	0.5705765407554672

Table 2: Accuracy Score

3.3 Confusion Matrix

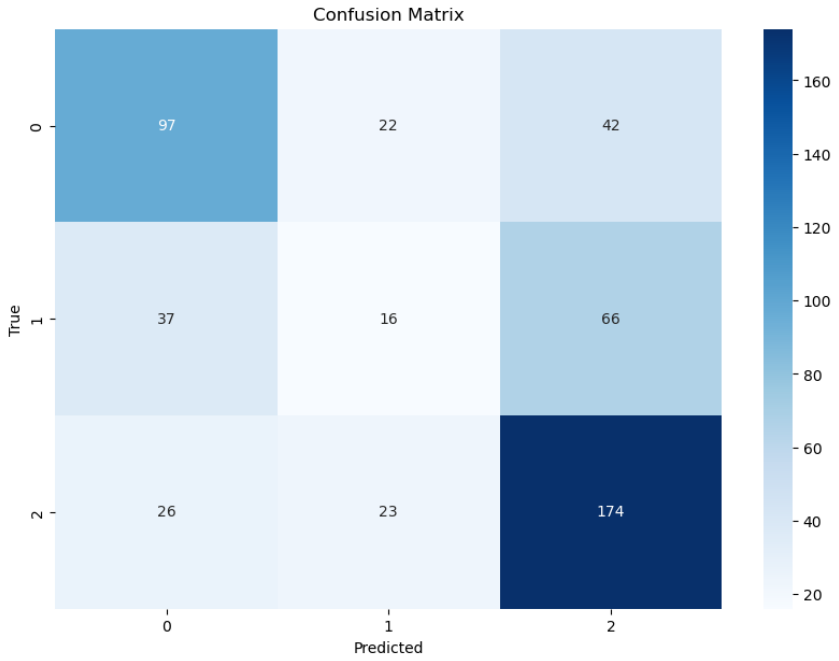

Figure 1: Confusion Matrix

3.4 Candlestick Chart with Predicted Trends

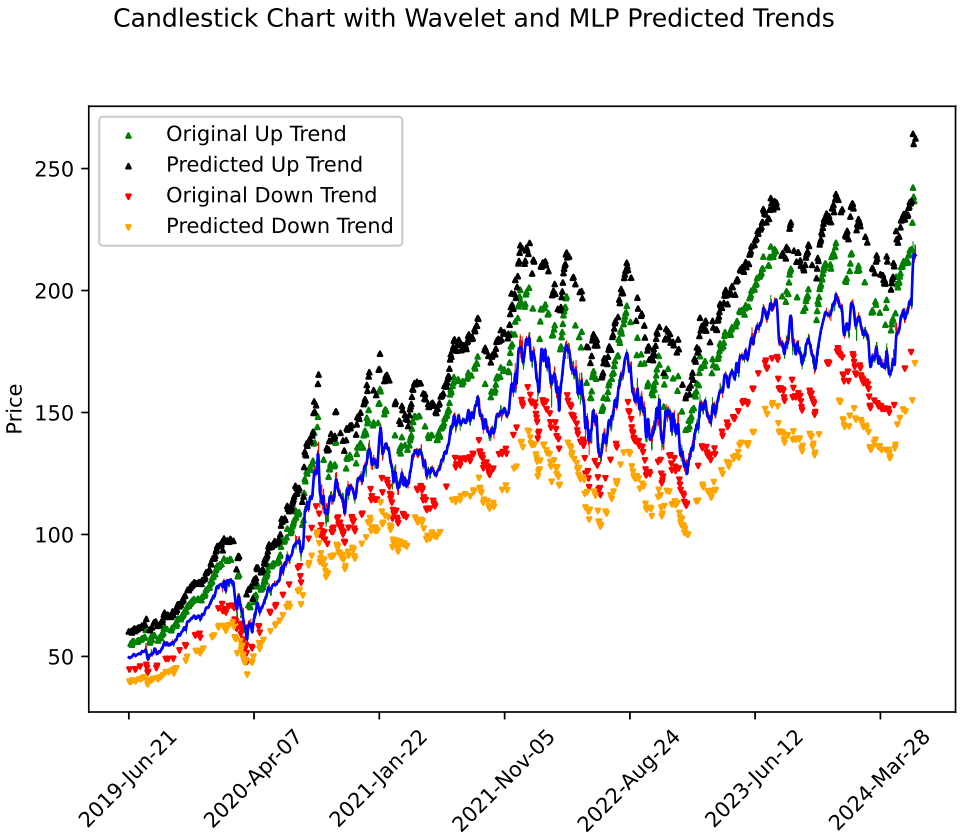

Figure 2: Candlestick Chart with Wavelet and MLP Predicted Trends

3.5 Discussion

The classification report (Table 1) shows that the model achieved a precision of 0.59 for the 'down' class, 0.30 for the 'sideways' class, and 0.63 for the 'up' class. The recall values are 0.58, 0.18, and 0.77 respectively. The overall accuracy of the model is 0.57, as indicated in Table 2.

The confusion matrix (Figure 1) provides a visual representation of the model's performance, showing the number of correct and incorrect predictions for each class.

The candlestick chart (Figure 2) illustrates the predicted trends over time, with arrows indicating the original and predicted trends for 'up' and 'down' movements.

The results indicate that while the model performs reasonably well for predicting 'up' and 'down' trends, it struggles with the 'sideways' trend, as evidenced by the lower precision and recall values for this class. This could be due to the inherent difficulty in distinguishing 'sideways' trends, which often exhibit less distinct patterns compared to 'up' and 'down' trends.

4 Conclusion

The model demonstrates a method for predicting stock price trends using historical data, wavelet transformation, and neural network classification. The overall accuracy and

performance metrics suggest that the model is effective in identifying 'up' and 'down' trends but may require further refinement for 'sideways' trends. Future work may involve optimizing the model parameters, exploring additional features, and incorporating more advanced techniques to improve prediction accuracy.