Assignment 2: WiDS Kalman Filtered Trend Trader

Manthan
Roll No: 24B2511

1 Introduction

Financial markets exhibit non-stationary dynamics where relationships between price, momentum, and volatility evolve over time. Traditional static models often fail to adapt to such time-varying behavior. This report details a trading strategy for Microsoft Corp. (MSFT) that combines feature-based machine learning with Kalman filtering to estimate latent parameters governing the stock's price.

2 Methodology and Model Formulation

2.1 Data Acquisition and Feature Engineering

Daily historical data for MSFT was fetched from Yahoo Finance (2015–2024). To capture market dynamics, the following features were engineered:

- **Trend:** 5-day and 20-day Moving Averages (MA).
- **Momentum:** Log returns, lagged returns, and Rate of Change (ROC).
- **Risk:** Rolling volatility measures based on a 20-day window.
- **Volume:** Volume-based change indicators.

2.2 Kalman Filter State-Space Representation

We formulate a state-space model where the latent state represents time-varying regression coefficients.

- **State Vector (β_t):** Represents the time-varying intercept (α) and slope (β) relating the moving average to the price.
- **Observation Equation:** $P_t = H_t\beta_t + v_t$, where $v_t \sim N(0, R)$.
- **Transition Equation:** $\beta_t = \beta_{2-1} + w_t$, where $w_t \sim N(0, Q)$.

2.3 Machine Learning and Strategy Logic

The Kalman-filtered states are used as inputs for a tree-based Machine Learning model to predict future price ratios.

- **Buy Signal:** Generated if the predicted ratio is significantly higher than the current ratio.
- **Sell Signal:** Generated if the predicted ratio is significantly lower than the current ratio.

3 Python Implementation

The following code implements the full pipeline, including data fetching, Kalman filtering, ML prediction, and backtesting.

```
import yfinance as yf
import pandas as pd
import numpy as np
from pykalman import KalmanFilter
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt

def get_msft_data():
    df = yf.download("MSFT", start="2015-01-01", end="2024-12-31")
    return df

def engineer_features(df):
    df['MA5'] = df['Close'].rolling(window=5).mean()
    df['MA20'] = df['Close'].rolling(window=20).mean()
    df['Log_Ret'] = np.log(df['Close'] / df['Close'].shift(1))
    df['ROC'] = (df['Close'] - df['Close'].shift(5)) / df['Close'].shift(5)
    df['Vol'] = df['Log_Ret'].rolling(window=20).std()
    return df.dropna()

def apply_kalman_filter(df):
    obs_values = df['Close'].values
    features = df['MA20'].values
    obs_mat = np.expand_dims(np.column_stack([np.ones(len(features)), features]),
    axis=1)

    kf = KalmanFilter(
        n_dim_obs=1, n_dim_state=2,
        initial_state_mean=[0, 1],
        transition_matrices=np.eye(2),
        observation_matrices=obs_mat,
        observation_covariance=1.0,
        transition_covariance=1e-4 * np.eye(2)
    )

    state_means, _ = kf.filter(obs_values)
    df['KF_Intercept'] = state_means[:, 0]
    df['KF_Slope'] = state_means[:, 1]
    return df

def generate_signals(df):
    df['Target'] = df['Close'].shift(-1) / df['Close']
    df = df.dropna().copy()

    X = df[['KF_Intercept', 'KF_Slope', 'ROC', 'Vol']]
    y = df['Target']

    split = int(len(df) * 0.8)
    X_train, X_test = X[:split], X[split:]
    y_train, y_test = y[:split], y[split:]

    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    df.loc[X_test.index, 'Pred_Ratio'] = model.predict(X_test)

    threshold = 0.001
    df['Signal'] = 0
    df.loc[df['Pred_Ratio'] > (1 + threshold), 'Signal'] = 1
    df.loc[df['Pred_Ratio'] < (1 - threshold), 'Signal'] = -1

```

```
    return df.iloc[split:].copy()

def backtest(df):
    cost = 0.0005
    df['Strat_Ret'] = df['Signal'].shift(1) * df['Log_Ret']
    trades = df['Signal'].diff().fillna(0).abs()
    df['Strat_Ret'] -= (trades * cost)

    df['Equity_Curve'] = df['Strat_Ret'].cumsum().apply(np.exp)
    df['Hold_Curve'] = df['Log_Ret'].cumsum().apply(np.exp)
    return df

data = get_msft_data()
data = engineer_features(data)
data = apply_kalman_filter(data)
trades = generate_signals(data)
results = backtest(trades)
```

Listing 1: Complete Trading Strategy Source Code

4 Strategy Visualization

Visualizing the equity curve is essential for understanding the path-dependency of returns and identifying periods of drawdown.

```
def plot_performance(results):
    plt.figure(figsize=(12, 6))
    plt.plot(results['Equity_Curve'], label='Kalman-ML Strategy', color='black', linewidth=1.5)
    plt.plot(results['Hold_Curve'], label='MSFT Buy & Hold', color='gray', linestyle='--')
    plt.title('Strategy Cumulative Returns vs Benchmark')
    plt.xlabel('Date')
    plt.ylabel('Cumulative Return')
    plt.legend()
    plt.grid(True, linestyle='--', linewidth=0.5)
    plt.show()

plot_performance(results)
```

Listing 2: Equity Curve Generation

5 Performance Evaluation

The strategy performance is evaluated using the following metrics:

- **Cumulative Return:** Total profit generated over the test period.
- **Sharpe Ratio:** Risk-adjusted return calculation.
- **Maximum Drawdown:** Maximum observed loss from a peak.

6 Conclusion

The Kalman-ML integration effectively adapts to time-varying market parameters. By using non-causal features and accounting for transaction costs, the strategy provides a realistic framework for algorithmic trading in volatile environments.