momentum_ML_final

April 13, 2025

1 Machine Learning Momentum Strategy

[11]:
```
import os
import time
import logging
from datetime import datetime, timedelta
import pandas as pd
import numpy as np
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import MarketOrderRequest
from alpaca.trading.enums import OrderSide, TimeInForce, AssetClass
from alpaca.data.historical import StockHistoricalDataClient
from alpaca.data.requests import StockBarsRequest
from alpaca.data.timeframe import TimeFrame
from sklearn.ensemble import RandomForestClassifier
from pathlib import Path
from logger_setup import get_bot_logger
```

[12]:
```
from dotenv import load_dotenv

load_dotenv()
```

[12]:
```
True
```

[13]:
```
BOT_NAME = "momentum_ml_carlo"
log_dir = Path.cwd() / BOT_NAME
log_dir.mkdir(parents=True, exist_ok=True)
logger = get_bot_logger(BOT_NAME, f"{Path.cwd()}/{BOT_NAME}")

class MomentumStrategy:
    """
    A trading bot implementing an optimized momentum strategy using the Alpaca␣
 ↪API.
    This version uses a normalized momentum factor, corrective AI, and adaptive␣
 ↪stop loss.
    """
    def __init__(self):
```

```
        """Initialize the momentum strategy bot with API credentials and␣
↪settings."""
        self.api_key = os.environ[f"{BOT_NAME}_ALPACA_API_KEY"]
        self.api_secret = os.environ[f"{BOT_NAME}_ALPACA_API_SECRET"]
        if not self.api_key or not self.api_secret:
            raise ValueError("API key and secret must be provided in␣
↪environment variables")

        # Initialize Alpaca clients
        self.trading_client = TradingClient(self.api_key, self.api_secret,␣
↪paper=True)
        self.data_client = StockHistoricalDataClient(self.api_key, self.
↪api_secret)

        # Trading parameters
        self.symbols = ["GOOGL", "AAPL", "AMZN", "META", "MSFT", "NVDA"]
        self.timeframe = TimeFrame.Day
        # Use a longer lookback period to ensure enough data for the 252-day␣
↪window.
        self.lookback_days = 500
        # Optimized parameters from backtesting:
        self.momentum_threshold = 2.0    # Normalized momentum threshold
        self.stop_loss_multiplier = 1.1     # Adaptive stop loss multiplier
        # Random Forest parameters for corrective AI:
        self.rf_params = {'n_estimators': 150, 'max_depth': 15}

        # Long and short Probabilities
        self.long_prob = 0.70
        self.short_prob = 0.25

        logger.info(f"Optimized Momentum strategy bot {BOT_NAME} initialized␣
↪with {len(self.symbols)} symbols")

    def is_market_open(self):
        """Return True if the market is open, using Alpaca's market clock."""
        clock = self.trading_client.get_clock()
        return clock.is_open

    def get_account_info(self):
        """Retrieve account information and log details."""
        account = self.trading_client.get_account()
        logger.info(f"Account ID: {account.id}")
        logger.info(f"Cash: ${account.cash}")
        logger.info(f"Portfolio value: ${account.portfolio_value}")
        logger.info(f"Buying power: ${account.buying_power}")
        return account
```

```
    def get_positions(self):
        """Retrieve current positions and return as a dict keyed by symbol."""
        positions = self.trading_client.get_all_positions()
        pos_dict = {}
        for position in positions:
            logger.info(f"Position: {position.symbol}, Qty: {position.qty},␣
↪Market Value: ${position.market_value}")
            pos_dict[position.symbol] = position
        return pos_dict

    def get_historical_data(self, symbol, days=None):
        """Fetch historical daily bar data for a given symbol."""
        if days is None:
            days = self.lookback_days
        end = datetime.now()
        start = end - timedelta(days=days)

        request_params = StockBarsRequest(
            symbol_or_symbols=symbol,
            timeframe=self.timeframe,
            start=start,
            end=end
        )
        bars = self.data_client.get_stock_bars(request_params)
        df = bars.df
        if df.empty:
            logger.warning(f"No data found for {symbol}")
            return None
        df = df.reset_index().sort_values(by=["timestamp"])
        logger.info(f"Retrieved {len(df)} bars for {symbol}")
        return df

    def calculate_normalized_momentum(self, df):
        """
        Calculate a normalized momentum factor over a 252-day window:
          factor = [((close[-21] - close[-252]) / close[-252])
                    - ((close[-1] - close[-21]) / close[-21])] / std(returns␣
↪over last 126 days)
        """
        if df is None or len(df) < 252:
            return None
        window = df['close'].iloc[-252:]
        returns = window.pct_change().iloc[-126:]
        std_returns = np.std(returns)
        if std_returns == 0 or np.isnan(std_returns):
            return None
```

```
        long_term = (window.iloc[-21] - window.iloc[0]) / window.iloc[0]
        short_term = (window.iloc[-1] - window.iloc[-21]) / window.iloc[-21]
        factor = (long_term - short_term) / std_returns
        logger.info(f"Calculated normalized momentum: {factor:.2f}")
        return factor

    def compute_features_for_ai(self, df):
        """
        Build a feature set for corrective AI using the latest data.
        Features: normalized momentum, 10-day volatility, RSI.
        """
        norm_mom = self.calculate_normalized_momentum(df)
        df['return'] = df['close'].pct_change()
        vol_10 = df['return'].rolling(10).std().iloc[-1]
        # Simple RSI calculation:
        delta = df['close'].diff()
        gain = delta.clip(lower=0)
        loss = -delta.clip(upper=0)
        avg_gain = gain.rolling(14).mean().iloc[-1]
        avg_loss = loss.rolling(14).mean().iloc[-1]
        if avg_loss == 0:
            rsi_val = 100
        else:
            rs = avg_gain / avg_loss
            rsi_val = 100 - (100 / (1 + rs))
        features = {
            'momentum': norm_mom,
            'volatility': vol_10,
            'rsi': rsi_val
        }
        return features

    def train_corrective_ai(self, df):
        """
        Train a RandomForest classifier using historical features and labels.
        For each valid window, compute:
          - Features: normalized momentum, 10-day volatility, RSI.
          - Label: 5-day forward return > 0 (1) or not (0).
        Returns the trained model.
        """
        if len(df) < 300:
            return None
        features_list = []
        labels_list = []
        # Iterate over rows where a full 252-day window and 5-day lookahead are␣
↪available.
        for i in range(252, len(df) - 5):
```

```
            window = df['close'].iloc[i - 252:i]
            returns = window.pct_change().iloc[-126:]
            std_returns = np.std(returns)

            if std_returns == 0 or np.isnan(std_returns):
                continue

            long_term = (window.iloc[-21] - window.iloc[0]) / window.iloc[0]
            short_term = (window.iloc[-1] - window.iloc[-21]) / window.iloc[-21]
            mom = (long_term - short_term) / std_returns
            vol = df['close'].pct_change().iloc[i - 10:i].std()
            delta = df['close'].diff().iloc[i - 14:i]
            gain = delta.clip(lower=0)
            loss = -delta.clip(upper=0)
            avg_gain = gain.mean()
            avg_loss = loss.mean()
            rsi_val = 100 if avg_loss == 0 else 100 - (100 / (1 + avg_gain /␣
↪avg_loss))
            features_list.append([mom, vol, rsi_val])
            future_return = (df['close'].iloc[i + 5] - df['close'].iloc[i]) /␣
↪df['close'].iloc[i]
            labels_list.append(1 if future_return > 0 else 0)
        if len(features_list) < 50:
            return None
        X = pd.DataFrame(features_list, columns=['momentum', 'volatility',␣
↪'rsi'])
        y = pd.Series(labels_list)
        model = RandomForestClassifier(**self.rf_params, random_state=42)
        model.fit(X, y)
        return model

    def run_strategy(self):
        """Run the optimized momentum strategy using live Alpaca data."""
        logger.info("Running optimized momentum strategy...")
        if not self.is_market_open():
            logger.info("Market is closed according to Alpaca. Running in␣
↪simulation mode.")
        else:
            logger.info("Market is open.")

        positions = self.get_positions()
        account = self.get_account_info()
        total_cash = float(account.cash) * 0.9
        allocation_per_stock = total_cash / len(self.symbols)

        for symbol in self.symbols:
            logger.info(f"Analyzing {symbol}...")
```

```
            df = self.get_historical_data(symbol, days=self.lookback_days)
            if df is None or len(df) < 252:
                logger.warning(f"Not enough data for {symbol}")
                continue

            norm_momentum = self.calculate_normalized_momentum(df)
            if norm_momentum is None:
                continue

            # Train corrective AI on historical data for this symbol.
            model = self.train_corrective_ai(df)
            if model is None:
                logger.warning(f"Not enough data to train corrective AI for␣
↪{symbol}")
                continue

            # Get latest features and predicted probability.
            features = self.compute_features_for_ai(df)
            X_latest = pd.DataFrame([features])
            prob = model.predict_proba(X_latest)[0][1]  # probability of␣
↪positive outcome
            logger.info(f"Corrective AI probability for {symbol}: {prob:.2f}")

            # Determine signals:
            signal_long = norm_momentum > self.momentum_threshold and prob >␣
↪self.long_prob
            signal_short = norm_momentum < -self.momentum_threshold and prob <␣
↪self.short_prob

            current_price = df['close'].iloc[-1]
            qty_for_trade = int(allocation_per_stock // current_price)
            if qty_for_trade <= 0:
                logger.warning(f"Not enough allocated cash to trade {symbol} at␣
↪${current_price:.2f}")
                continue

            # Adaptive Stop Loss: compute 20-day volatility and today's return.
            df['return'] = df['close'].pct_change()
            vol_20 = df['return'].rolling(20).std().iloc[-1]
            if len(df) < 2:
                continue
            today_return = (df['close'].iloc[-1] - df['close'].iloc[-2]) /␣
↪df['close'].iloc[-2]
            stop_loss_triggered = False
            if symbol in positions:
                pos = positions[symbol]
```

```
                if float(pos.qty) > 0 and today_return < -self.
↪stop_loss_multiplier * vol_20:
                    stop_loss_triggered = True
                    logger.info(f"Stop loss triggered for {symbol} LONG:␣
↪today's return {today_return:.2%} < -{self.stop_loss_multiplier} * {vol_20:.
↪2%}")
                elif float(pos.qty) < 0 and today_return > self.
↪stop_loss_multiplier * vol_20:
                    stop_loss_triggered = True
                    logger.info(f"Stop loss triggered for {symbol} SHORT:␣
↪today's return {today_return:.2%} > {self.stop_loss_multiplier} * {vol_20:.
↪2%}")

            # If already in a position, check if we need to exit.
            if symbol in positions:
                pos = positions[symbol]
                pos_qty = float(pos.qty)
                if pos_qty > 0 and (norm_momentum <= self.momentum_threshold or␣
↪stop_loss_triggered):
                    logger.info(f"Liquidating LONG position for {symbol} due to␣
↪momentum reversal or stop loss.")
                    self.submit_order(symbol, OrderSide.SELL, pos_qty)
                elif pos_qty < 0 and (norm_momentum >= -self.momentum_threshold␣
↪or stop_loss_triggered):
                    logger.info(f"Covering SHORT position for {symbol} due to␣
↪momentum reversal or stop loss.")
                    self.submit_order(symbol, OrderSide.BUY, abs(pos_qty))
                positions = self.get_positions()  # Refresh positions after exit

            # Entry logic: if no position exists, enter trade based on signals.
            if symbol not in positions:
                if signal_long:
                    logger.info(f"BUY signal for {symbol}: normalized momentum␣
↪{norm_momentum:.2f} exceeds threshold and probability {prob:.2f} > 0.6.␣
↪Buying {qty_for_trade} shares at ${current_price:.2f}")
                    self.submit_order(symbol, OrderSide.BUY, qty_for_trade)
                elif signal_short:
                    logger.info(f"SHORT signal for {symbol}: normalized␣
↪momentum {norm_momentum:.2f} below negative threshold and probability {prob:.
↪2f} < 0.4. Shorting {qty_for_trade} shares at ${current_price:.2f}")
                    self.submit_order(symbol, OrderSide.SELL, qty_for_trade)
                else:
                    logger.info(f"No valid trade signal for {symbol}:␣
↪normalized momentum {norm_momentum:.2f}, probability {prob:.2f}")

        logger.info("Optimized momentum strategy execution completed.")
```

```
    def submit_order(self, symbol, side, qty):
        """Submit a market order for the given symbol using Alpaca API."""
        try:
            order_data = MarketOrderRequest(
                symbol=symbol,
                qty=qty,
                side=side,
                time_in_force=TimeInForce.DAY
            )
            order = self.trading_client.submit_order(order_data)
            logger.info(f"Order placed: {side} {qty} shares of {symbol}. Order␣
 ↪ID: {order.id}")
            return order
        except Exception as e:
            logger.error(f"Error submitting order for {symbol}: {e}")
            return None

def main():
    logger.info(f"Starting {BOT_NAME}")
    try:
        bot = MomentumStrategy()
        bot.run_strategy()
        logger.info(f"{BOT_NAME} completed successfully")
    except Exception as e:
        logger.error(f"Error running {BOT_NAME}: {e}", exc_info=True)
        raise

if __name__ == "__main__":
    main()
```

```
2025-04-06 00:26:25,024 - momentum_ML - INFO - Logger initialized and header
written.
2025-04-06 00:26:25,029 - momentum_ML - INFO - Starting momentum_ML
2025-04-06 00:26:25,031 - momentum_ML - INFO - Optimized Momentum strategy bot
momentum_ML initialized with 6 symbols
2025-04-06 00:26:25,032 - momentum_ML - INFO - Running optimized momentum
strategy…
2025-04-06 00:26:25,345 - momentum_ML - INFO - Market is closed according to
Alpaca. Running in simulation mode.
2025-04-06 00:26:25,428 - momentum_ML - INFO - Position: AMZN, Qty: 83, Market
Value: $14193
2025-04-06 00:26:25,429 - momentum_ML - INFO - Position: GOOGL, Qty: 99, Market
Value: $14414.4
2025-04-06 00:26:25,507 - momentum_ML - INFO - Account ID: 1a9647ec-
da17-4d78-b173-d7cc4ced6608
2025-04-06 00:26:25,508 - momentum_ML - INFO - Cash: $70077.25
2025-04-06 00:26:25,508 - momentum_ML - INFO - Portfolio value: $98684.65
```

```
2025-04-06 00:26:25,509 - momentum_ML - INFO - Buying power: $168761.9
2025-04-06 00:26:25,509 - momentum_ML - INFO - Analyzing GOOGL…
2025-04-06 00:26:25,837 - momentum_ML - INFO - Retrieved 341 bars for GOOGL
2025-04-06 00:26:25,841 - momentum_ML - INFO - Calculated normalized momentum:
15.86
2025-04-06 00:26:34,376 - momentum_ML - INFO - Calculated normalized momentum:
15.86
2025-04-06 00:26:34,396 - momentum_ML - INFO - Corrective AI probability for
GOOGL: 0.41
2025-04-06 00:26:34,399 - momentum_ML - INFO - Stop loss triggered for GOOGL
LONG: today's return -3.40% < -1.1 * 2.37%
2025-04-06 00:26:34,399 - momentum_ML - INFO - Liquidating LONG position for
GOOGL due to momentum reversal or stop loss.
2025-04-06 00:26:34,563 - momentum_ML - INFO - Order placed: OrderSide.SELL 99.0
shares of GOOGL. Order ID: d66d10a3-5e7c-4ee8-a940-a18863455947
2025-04-06 00:26:34,642 - momentum_ML - INFO - Position: AMZN, Qty: 83, Market
Value: $14193
2025-04-06 00:26:34,642 - momentum_ML - INFO - Position: GOOGL, Qty: 99, Market
Value: $14414.4
2025-04-06 00:26:34,643 - momentum_ML - INFO - Analyzing AAPL…
2025-04-06 00:26:34,812 - momentum_ML - INFO - Retrieved 341 bars for AAPL
2025-04-06 00:26:34,814 - momentum_ML - INFO - Calculated normalized momentum:
35.00
2025-04-06 00:26:35,191 - momentum_ML - INFO - Calculated normalized momentum:
35.00
2025-04-06 00:26:35,210 - momentum_ML - INFO - Corrective AI probability for
AAPL: 0.43
2025-04-06 00:26:35,212 - momentum_ML - INFO - No valid trade signal for AAPL:
normalized momentum 35.00, probability 0.43
2025-04-06 00:26:35,213 - momentum_ML - INFO - Analyzing AMZN…
2025-04-06 00:26:35,376 - momentum_ML - INFO - Retrieved 341 bars for AMZN
2025-04-06 00:26:35,378 - momentum_ML - INFO - Calculated normalized momentum:
12.27
2025-04-06 00:26:35,861 - momentum_ML - INFO - Calculated normalized momentum:
12.27
2025-04-06 00:26:35,880 - momentum_ML - INFO - Corrective AI probability for
AMZN: 0.60
2025-04-06 00:26:35,882 - momentum_ML - INFO - Stop loss triggered for AMZN
LONG: today's return -4.15% < -1.1 * 2.87%
2025-04-06 00:26:35,883 - momentum_ML - INFO - Liquidating LONG position for
AMZN due to momentum reversal or stop loss.
2025-04-06 00:26:35,965 - momentum_ML - INFO - Order placed: OrderSide.SELL 83.0
shares of AMZN. Order ID: 5a8dd446-92e4-48e7-a78e-67e7cdadd077
2025-04-06 00:26:36,041 - momentum_ML - INFO - Position: AMZN, Qty: 83, Market
Value: $14193
2025-04-06 00:26:36,041 - momentum_ML - INFO - Position: GOOGL, Qty: 99, Market
Value: $14414.4
2025-04-06 00:26:36,042 - momentum_ML - INFO - Analyzing META…
```

```
2025-04-06 00:26:36,208 - momentum_ML - INFO - Retrieved 341 bars for META
2025-04-06 00:26:36,210 - momentum_ML - INFO - Calculated normalized momentum:
20.14
2025-04-06 00:26:36,593 - momentum_ML - INFO - Calculated normalized momentum:
20.14
2025-04-06 00:26:36,613 - momentum_ML - INFO - Corrective AI probability for
META: 0.40
2025-04-06 00:26:36,616 - momentum_ML - INFO - No valid trade signal for META:
normalized momentum 20.14, probability 0.40
2025-04-06 00:26:36,616 - momentum_ML - INFO - Analyzing MSFT…
2025-04-06 00:26:36,793 - momentum_ML - INFO - Retrieved 341 bars for MSFT
2025-04-06 00:26:36,800 - momentum_ML - INFO - Calculated normalized momentum:
1.75
2025-04-06 00:26:37,283 - momentum_ML - INFO - Calculated normalized momentum:
1.75
2025-04-06 00:26:37,301 - momentum_ML - INFO - Corrective AI probability for
MSFT: 0.51
2025-04-06 00:26:37,303 - momentum_ML - INFO - No valid trade signal for MSFT:
normalized momentum 1.75, probability 0.51
2025-04-06 00:26:37,304 - momentum_ML - INFO - Analyzing NVDA…
2025-04-06 00:26:37,467 - momentum_ML - INFO - Retrieved 341 bars for NVDA
2025-04-06 00:26:37,468 - momentum_ML - INFO - Calculated normalized momentum:
-20.50
2025-04-06 00:26:37,849 - momentum_ML - INFO - Calculated normalized momentum:
-20.50
2025-04-06 00:26:37,866 - momentum_ML - INFO - Corrective AI probability for
NVDA: 0.44
2025-04-06 00:26:37,868 - momentum_ML - INFO - No valid trade signal for NVDA:
normalized momentum -20.50, probability 0.44
2025-04-06 00:26:37,869 - momentum_ML - INFO - Optimized momentum strategy
execution completed.
2025-04-06 00:26:37,870 - momentum_ML - INFO - momentum_ML completed
successfully
```

[ ]:

[ ]: