Untitled1

April 4, 2023

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from qiskit_machine_learning.neural_networks import CircuitQNN

# Custom functions
def preprocess_data(data):
    # Extract relevant features from the dataset
    features = data[['open', 'high', 'low', 'close', 'volume',␣
 ↪'implied_volatility']]

    # Calculate additional features if necessary (e.g., technical indicators)
    # features['sma'] = data['close'].rolling(window=10).mean()

    # Set the target variable (e.g., next day's close)
    target = data['close'].shift(-1)

    # Remove rows with missing values
    features.dropna(inplace=True)
    target.dropna(inplace=True)

    return features, target

def preprocess_qnn_data(qnn_data):
    # Extract relevant features from the dataset
    qnn_features = qnn_data[['volume', 'strike']]

    # Calculate additional features if necessary
    # qnn_features['time_to_expiry'] = ...

    # Set the target variable (e.g., intraday volume spikes)
    qnn_target = qnn_data['volume_spike']
```

```python
    # Remove rows with missing values
    qnn_features.dropna(inplace=True)
    qnn_target.dropna(inplace=True)

    return qnn_features, qnn_target

from sklearn.metrics import mean_squared_error, accuracy_score

def evaluate_qnn_performance(qnn_predictions, qnn_y_test):
    # Calculate the Mean Squared Error (MSE) between the predictions and the actual values
    mse = mean_squared_error(qnn_y_test, qnn_predictions)
    print(f'Mean Squared Error: {mse}')

    # Calculate the accuracy of the QNN model's predictions (assumes binary classification)
    accuracy = accuracy_score(qnn_y_test, np.round(qnn_predictions))
    print(f'Accuracy: {accuracy}')

def determine_optimal_trade_times(model, qnn, current_market_data):
    # Determine the optimal time to enter the trade based on QNN predictions
    enter_trade_time = 'market_open' if qnn.predict(current_market_data) > 0.5 else '11_am'

    # Determine the optimal time to exit the trade based on other factors
    # You can replace this with more sophisticated logic based on your strategy
    exit_trade_time = 'peak_retail_close' if current_market_data['exit_indicator'] > 0.5 else 'algo_dump_2_pm'

    return enter_trade_time, exit_trade_time

def execute_trade(atm_options, otm_options, enter_trade_time, exit_trade_time):
    # Implement the logic to execute the trade based on the given options contracts and optimal times
    print(f'Entering trade at {enter_trade_time} with ATM options: {atm_options} and OTM options: {otm_options}')
    print(f'Exiting trade at {exit_trade_time}')

    # Execute the trade using your preferred trading platform/API
    # This part depends on the specific trading platform or API you're using


# Load historical SPX options data
data = pd.read_csv('spx_options_historical_data.csv')
```

```python
# Preprocess the data and extract features (e.g., market price, implied␣
↪volatility, etc.)
features, target = preprocess_data(data)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target,␣
↪test_size=0.2, random_state=42)

# Scale the features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define the neural network model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(1)
])

# Compile the model
model.compile(optimizer=Adam(lr=0.001), loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.1,␣
↪verbose=1)

# Evaluate the model's performance on the test data
mse = model.evaluate(X_test, y_test)
print(f'Mean Squared Error: {mse}')

# Load historical data for the QNN model
qnn_data = pd.read_csv('qnn_spx_historical_data.csv')

# Preprocess the data and extract features (e.g., volume, strikes, etc.)
qnn_features, qnn_target = preprocess_qnn_data(qnn_data)

# Split the data into training and testing sets for the QNN model
qnn_X_train, qnn_X_test, qnn_y_train, qnn_y_test =␣
↪train_test_split(qnn_features, qnn_target, test_size=0.2, random_state=42)

# Define the QNN architecture (modify as needed)
qnn_architecture = {
    'num_qubits': 10,
```

```
    'reps': 2,
    'entanglement': 'linear',
    'input_mapping': 'encode',
}

# Create the QNN
qnn = CircuitQNN(**qnn_architecture)

# Train the QNN on the training data
qnn.fit(qnn_X_train, qnn_y_train)


# Make predictions on the test data
qnn_predictions = qnn.predict(qnn_X_test)

# Evaluate the QNN's performance and adjust hyperparameters as necessary
evaluate_qnn_performance(qnn_predictions, qnn_y_test)

# Get current market data
current_market_data = get_current_market_data()

# Use the deep learning model to choose the appropriate SPX options contracts
scaled_current_market_data = scaler.transform(current_market_data)
atm_options, otm_options = select_options_contracts(model,
 ↪scaled_current_market_data)

# Determine the optimal time to enter and exit the trade
enter_trade_time, exit_trade_time = determine_optimal_trade_times(model, qnn,
 ↪current_market_data)

# Use the determined optimal times to enter and exit the trade with the
 ↪selected options contracts
execute_trade(atm_options, otm_options, enter_trade_time, exit_trade_time)
```

1 This is the 2nd Strategy : In this strategy, we use a deep learning model to predict the direction of intraday price movements. The model is then used to select appropriate options contracts based on the predicted price direction. We also implement a stop-loss and take-profit approach to manage risk.

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

def preprocess_data(data):
    # Extract relevant features from the dataset
    features = data[['open', 'high', 'low', 'close', 'volume',␣
↪'implied_volatility']]

    # Calculate additional features if necessary (e.g., technical indicators)
    # features['sma'] = data['close'].rolling(window=10).mean()

    # Set the target variable (e.g., next day's close)
    target = data['close'].shift(-1)

    # Remove rows with missing values
    features.dropna(inplace=True)
    target.dropna(inplace=True)

    return features, target

def select_options_contracts(model, current_market_data):
    # Scale the current market data
    scaled_current_market_data = scaler.transform(current_market_data)

    # Use the deep learning model to predict the direction of the price movement
    predicted_direction = model.predict(scaled_current_market_data)

    # Choose the appropriate options contracts based on the predicted price␣
↪direction
    if predicted_direction > 0.5:
        option_contracts = "call_options"
    else:
        option_contracts = "put_options"

    return option_contracts, predicted_direction

def calculate_stop_loss_take_profit(entry_price, predicted_direction):
    # Calculate the stop-loss and take-profit levels based on the predicted␣
↪price direction
    stop_loss_percent = 0.02  # 2% stop-loss level
    take_profit_percent = 0.04  # 4% take-profit level

    if predicted_direction > 0.5:
        stop_loss = entry_price * (1 - stop_loss_percent)
        take_profit = entry_price * (1 + take_profit_percent)
    else:
```

```
        stop_loss = entry_price * (1 + stop_loss_percent)
        take_profit = entry_price * (1 - take_profit_percent)

    return stop_loss, take_profit

def execute_trade(option_contracts, entry_price, stop_loss, take_profit):
    # Implement the logic to execute the trade based on the given options␣
 ↪contracts and risk management parameters
    print(f'Entering trade with {option_contracts} at entry price:␣
 ↪{entry_price}, stop-loss: {stop_loss}, and take-profit: {take_profit}')

    # Execute the trade using your preferred trading platform/API



# Load historical SPX options data
data = pd.read_csv('spx_options_historical_data.csv')

# Preprocess the data and extract features (e.g., market price, implied␣
 ↪volatility, etc.)
features, target = preprocess_data(data)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target,␣
 ↪test_size=0.2, random_state=42)

# Scale the features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define the neural network model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer=Adam(lr=0.001), loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Train the model
```

```python
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.1,␣
 ↪verbose=1)

# Evaluate the model's performance on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_accuracy}')

# Get current market data
current_market_data = get_current_market_data()

# Use the deep learning model to select the appropriate options contracts based␣
 ↪on predicted price direction
option_contracts, predicted_direction = select_options_contracts(model,␣
 ↪current_market_data)

# Determine the entry price, stop-loss, and take-profit levels
entry_price = current_market_data['close']
stop_loss, take_profit = calculate_stop_loss_take_profit(entry_price,␣
 ↪predicted_direction)

# Execute the trade with the selected options contracts and specified risk␣
 ↪management parameters
execute_trade(option_contracts, entry_price, stop_loss, take_profit)
```