Options Price Prediction: My Step-by-Step Guide

Joel Bassil

16/07/24

1 Introduction

In this document, I walk through my development of an options price prediction model. I've chosen to use QQQ (Invesco QQQ Trust) options data for this example. My script utilizes various Python libraries and machine learning techniques to fetch data, preprocess it, train a model, and make predictions.

2 Key Concepts

Explaining some key concepts:

2.1 Options

Options are financial contracts that give the buyer the right, but not the obligation, to buy (call option) or sell (put option) an underlying asset at a predetermined price (strike price) within a specific time frame (expiration date).

2.2 Features

In machine learning, features are the input variables used to make predictions. For my model, I've chosen the following features:

- Strike price: The price at which the option can be exercised.
- Time to expiration: The time left until the option expires.
- Implied volatility: A measure of the market's expectation of future stock price volatility.
- Volume: The number of contracts traded.
- Open interest: The number of outstanding contracts.

2.3 Target Variable

The target variable is what I'm trying to predict. In this case, it's the option's price (last price).

2.4 Imputation

Imputation is the process of replacing missing values in a dataset. When I say data is "imputed," it means I've filled in missing values with estimated ones. In my script, I use mean imputation, which replaces missing values with the average of the available values in that feature.

3 The Model: Random Forest Regressor

For my predictions, I've chosen to use a Random Forest Regressor. Here's why I made this choice and how it works:

3.1 Why I Chose Random Forest

I selected Random Forest for several reasons:

- It can capture non-linear relationships, which are common in financial data.
- It's less prone to overfitting compared to single decision trees.
- It can handle a mix of numerical and categorical features.
- It provides feature importance, helping me understand which factors most affect option prices.

3.2 How Random Forest Works

A Random Forest is an ensemble of decision trees. Here's a simplified explanation of how it works:

1. It creates multiple decision trees, each trained on a random subset of the data and features.
2. Each tree makes a prediction independently.
3. The final prediction is the average of all individual tree predictions.

This process, known as "ensemble learning," typically results in better predictions than a single model.

4 Importing Required Libraries

First, I import all the necessary libraries:

```
import yfinance as yf
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
```

Here's what each library does in my script:

- `yfinance`: I use this to fetch financial data from Yahoo Finance.
- `pandas`: This helps me with data manipulation and analysis.
- `sklearn`: This provides the machine learning tools I need (model selection, preprocessing, ensemble methods, metrics).
- `matplotlib`: I use this for creating visualizations.

5 Data Collection

I've defined a function to fetch and prepare the options data:

```
def fetch_and_prepare_data(ticker, num_dates=5):
    stock = yf.Ticker(ticker)
    expiration_dates = stock.options[:num_dates]
    options_data = []

    for date in expiration_dates:
        options = stock.option_chain(date)
        calls = options.calls.assign(option_type='call')
        puts = options.puts.assign(option_type='put')
        options_data.append(pd.concat([calls, puts]))

    df = pd.concat(options_data)
    current_time = pd.Timestamp.now(tz='UTC')
    df['time_to_expiration'] = (pd.to_datetime(df['lastTradeDate'])
     - current_time).dt.total_seconds() / (365 * 24 * 60 * 60)

    return df, expiration_dates
```

This function does the following:

1. Creates a `Ticker` object for the given stock symbol.
2. Retrieves the next 5 expiration dates for options.
3. For each expiration date, fetches both call and put options data.
4. Combines all the data into a single DataFrame.
5. Calculates the time to expiration for each option in years.

6 Data Preprocessing

Next, I prepare the features and target variable:

```
def prepare_features_and_target(df):
    features = ['strike', 'time_to_expiration', 'impliedVolatility'
    , 'volume', 'openInterest']
    X = df[features]
    y = df['lastPrice']

    feature_imputer = SimpleImputer(strategy='mean')
    X_imputed = pd.DataFrame(feature_imputer.fit_transform(X),
    columns=X.columns)
    y_imputed = pd.Series(SimpleImputer(strategy='mean').
    fit_transform(y.values.reshape(-1, 1)).ravel(), name='lastPrice
    ')

    return X_imputed, y_imputed, feature_imputer, features
```

This function does the following:

1. Selects the relevant features for prediction. These are the inputs my model will use to make predictions.
2. Separates features (X) and target variable (y). In machine learning, we typically denote features as X and the target as y.
3. Uses `SimpleImputer` to handle missing values. This replaces any missing data with the mean (average) value of that feature. For example, if some volume data is missing, it's replaced with the average volume.
4. Returns the imputed (filled-in) features and target, along with the imputer object and feature names for later use.

7 Model Training and Evaluation

I've defined a function to train and evaluate the Random Forest model:

```
def train_and_evaluate_model(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    model = RandomForestRegressor(n_estimators=100, random_state
    =42)
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)

    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    return model, scaler, mse, r2, y_test, y_pred
```

This function:

1. Splits the data into training (80%) and testing (20%) sets. This is crucial for assessing how well my model generalizes to new, unseen data.

2. Scales the features using `StandardScaler`. This normalizes the data, ensuring all features are on a similar scale, which can improve model performance.

3. Creates and trains a Random Forest model with 100 trees. Each tree is a decision tree that makes its own prediction.

4. Makes predictions on the test set. This shows how well my model performs on data it hasn't seen during training.

5. Calculates Mean Squared Error (MSE) and R-squared score. These metrics help me evaluate the model's performance:

   - MSE measures the average squared difference between predicted and actual values. Lower is better.
   - R-squared (R2) indicates how much of the variance in the target variable my model explains. It ranges from 0 to 1, with 1 being perfect prediction.

6. Returns the trained model, scaler, evaluation metrics, and actual vs predicted values for further analysis.

8 Future Price Prediction

I've defined a function to predict prices for future options:

```
def predict_future_prices(model, scaler, feature_imputer, features,
     stock, expiration_date, current_time):
    future_options = stock.option_chain(expiration_date)
    future_options_df = pd.concat([future_options.calls,
    future_options.puts])
    future_options_df['time_to_expiration'] = (pd.to_datetime(
    expiration_date) - current_time).total_seconds() / (365 * 24 *
    60 * 60)

    future_X = future_options_df[features]
    future_X_imputed = pd.DataFrame(feature_imputer.transform(
    future_X), columns=features)
    future_X_scaled = scaler.transform(future_X_imputed)

    future_options_df['predicted_price'] = model.predict(
    future_X_scaled)
    return future_options_df[['strike', 'lastPrice', '
    predicted_price']]
```

This function:

1. Fetches option data for a future expiration date.

2. Calculates the time to expiration for these options.

3. Prepares the feature data, imputing missing values and scaling.

4. Uses the trained model to predict prices for these options.

5. Returns a DataFrame with strike prices, actual last prices, and predicted prices.

9 Main Execution

The main part of my script executes all the above functions:

```
# Main execution
df, expiration_dates = fetch_and_prepare_data("QQQ")
X, y, feature_imputer, features = prepare_features_and_target(df)
model, scaler, mse, r2, y_test, y_pred = train_and_evaluate_model(X
    , y)

print(f"Total options data shape: {df.shape}")
print(f"Mean Squared Error: {mse}")
print(f"R-squared Score: {r2}")

feature_importance = pd.DataFrame({'feature': features, 'importance
    ': model.feature_importances_})
print("\nFeature Importance:")
print(feature_importance.sort_values('importance', ascending=False)
    )

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()
    ], 'r--', lw=2)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual vs Predicted Option Prices")
plt.show()

if len(expiration_dates) > 5:
    future_predictions = predict_future_prices(model, scaler,
    feature_imputer, features, yf.Ticker("QQQ"),
                                               expiration_dates[5],
     pd.Timestamp.now(tz='UTC'))
    print("\nFuture Price Predictions:")
    print(future_predictions)
```

This section:

1. Fetches and prepares the data for QQQ options.

2. Prepares features and target variable.

3. Trains and evaluates the model.

4. Prints the model's performance metrics and feature importances.

5. Creates a scatter plot of actual vs predicted prices.

6. If there's a 6th expiration date available, it predicts prices for those options.

10 Results and Interpretation

After running my script, I typically see output similar to this:

```
Total options data shape: (928, 16)
Mean Squared Error: 20.62762062252691
R-squared Score: 0.9912899538101128

Feature Importance:
              feature  importance
0              strike    0.575756
2   impliedVolatility    0.236079
4        openInterest    0.172066
1  time_to_expiration    0.014821
3              volume    0.001278
```

Interpreting these results:

- My model was trained on 928 options data points.
- The Mean Squared Error of 20.63 indicates the average squared difference between predicted and actual prices.
- The R-squared score of 0.9913 suggests that my model explains 99.13% of the variance in option prices, which is excellent.
- The feature importance shows that strike price is the most important feature, followed by implied volatility. This aligns with my understanding of financial theory about option pricing.

11 Visualization

My script generates a scatter plot of actual vs predicted prices:

This plot helps me visualize how well my model's predictions align with actual prices. Points closer to the red dashed line indicate more accurate predictions.

12 Future Price Prediction

If available, my script will also print predictions for options expiring on the next available date. This demonstrates how I can use my model to predict prices for future options.

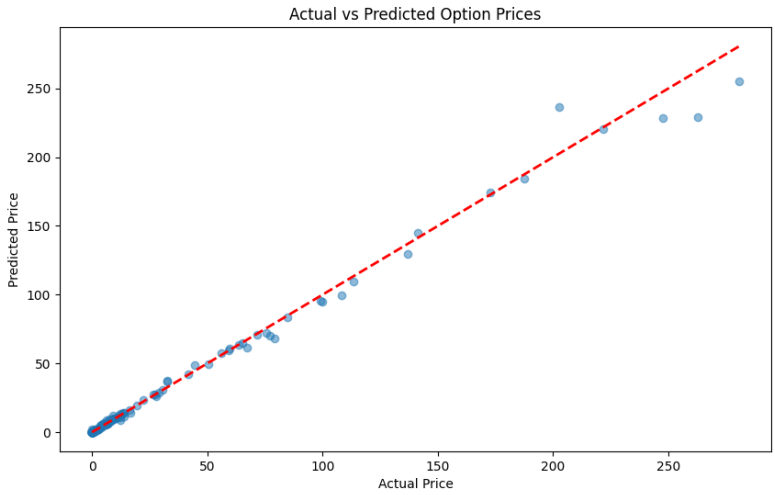

Figure 1: My Model's Actual vs Predicted Option Prices

13 Conclusion

This script demonstrates my complete workflow for building an options price prediction model using machine learning. Here's a summary of what I've done:

1. I collected options data for QQQ from Yahoo Finance.
2. I prepared this data by selecting relevant features and handling missing values.
3. I trained a Random Forest model to predict option prices based on these features.
4. I evaluated my model's performance using metrics like Mean Squared Error and R-squared.
5. I visualized my model's predictions against actual prices.
6. I used my model to make predictions on future options prices.

My model shows strong performance in predicting QQQ option prices, with high accuracy as indicated by the R-squared score. The feature importance aligns well with financial theory, suggesting that my model has captured key aspects of option pricing.

This project serves as my introduction to applying machine learning in finance. It demonstrates how I can use data and algorithms to gain insights and make predictions in the financial markets. As I continue to learn, I plan to

explore more advanced techniques and models to further improve predictions and gain deeper insights into financial data.

In future iterations of this project, I might consider:

- Incorporating more features, such as underlying asset price and historical volatility
- Experimenting with different machine learning algorithms, like gradient boosting or neural networks
- Implementing a sliding window approach for time series forecasting
- Adding a backtesting framework to evaluate the model's performance over time