Riskoptima Wealth Tech
Temmuz 2025

ML in Trading

Alper Duran Dinçer

Contents

Data Sources For Trading

Market Data

Price and volume information for financial instruments:

- OHLCV (Open, High, Low, Close, Volume)
- Tick data (trade-by-trade information)
- Order book data (bid/ask prices and volumes)
- Index values and constituents

Fundamental Data

Information about company financials and performance:

- Financial statements (income, balance sheet, cash flow)
- Earnings reports and forecasts
- Valuation metrics (P/E, P/B, EV/EBITDA)
- Dividend information

Alternative Data

Non-traditional data sources that may provide trading insights:

News sentiment and social media analysis

Satellite imagery (e.g., retail parking lots)

Credit card transaction data

Web traffic and app usage statistics

Derivatives Data

Information about options, futures, and other derivatives:

Options chains (strikes, expiries, premiums)

Implied volatility surfaces

Greeks (delta, gamma, theta, vega)

Open interest and volume

MARKET DATA

Market Data Sources

1 Yahoo Finance: Free historical data for stocks, ETFs, and indices

2 Alpha Vantage: API for real-time and historical data

3 Quandl: Financial, economic, and alternative datasets

4 IEX Cloud: Real-time and historical financial data

5 Polygon.io: Market data for stocks, options, forex, and crypto

Data Formats

- CSV: Simple tabular format for historical data
- JSON: Common format for API responses
- Parquet/HDF5: Efficient storage for large datase

```
# Fetching Stock Data with yfinance
import yfinance as yf
import pandas as pd

# Download historical data for AAPL
ticker = 'AAPL'
start_date = '2020-01-01'
end_date = '2023-12-31'

data = yf.download(ticker,
                   start=start_date,
                   end=end_date)

# Display the first 5 rows
print(data.head())

# Sample output:
# Open High Low Close Adj Close Volume
# Date
# 2020-01-02 74.059998 75.150002 73.797501 75.087502
```

OPTIONS DATA

Options Data Components

• Contract Specifications: Strike price, expiration date, option type (call/put)
• Pricing Information: Bid, ask, last price, mid price • Greeks: Delta, gamma, theta, vega, rho
• Implied Volatility: Market's expectation of future volatility
• Volume and Open Interest: Trading activity and outstanding contracts

```python
# Options Data Processing
import pandas as pd
import numpy as np
import bz2
import pickle

# Load compressed options data
def load_options_data(file_path):
    with bz2.BZ2File(file_path, 'rb') as f:
        data = pickle.load(f)
    return data

# File path
file_path =
'data_modules/spx_eom_expiry_options_2015_2022.bz2'

# Load the data
options_data = load_options_data(file_path)

# Convert to DataFrame
```

Sample Option Data

Strike	Expiry	Type	Last	Bid	Ask	IV	Delta
180	2023-07-21	Call	10.25	10.10	10.40	0.28	0.65
185	2023-07-21	Call	7.50	7.35	7.65	0.27	0.55
190	2023-07-21	Call	5.20	5.10	5.30	0.26	0.45

DATA STORAGE

Data Storage Methods

1. CSV Files
   - Simple text files with comma-separated values, widely used for tabular data.
   - Pros: ✓ Human-readable, ✓ Universal compatibility, ✓ Easy to share
   - Cons: ✗ Inefficient for large datasets, ✗ No schema enforcement, ✗ Slow to read/write

2. Compressed Files (bz2, gzip)
   - Compressed formats that reduce storage requirements for large datasets.
   - Pros: ✓ Reduced storage space, ✓ Faster network transfers, ✓ Maintains original format
   - Cons: ✗ Compression/decompression overhead, ✗ Not directly queryable, ✗ Requires full decompression

3. Pickle Files
   - Python-specific binary format for serializing Python objects.
   - Pros: ✓ Preserves Python objects, ✓ Fast for Python workflows, ✓ Maintains complex structures
   - Cons: ✗ Python-specific, ✗ Security concerns, ✗ Version compatibility issues

4. Parquet / HDF5
   - Columnar storage formats optimized for analytics and big data.
   - Pros: ✓ High compression, ✓ Column-oriented for analytics, ✓ Schema enforcement
   - Cons: ✗ Less human-readable, ✗ Requires specialized tools

```python
# Working with Different Storage Formats
import pandas as pd
import numpy as np
import pickle
import bz2

# Sample data
data = {
    'date': pd.date_range('2022-01-01', periods=100),
    'price': np.random.randn(100).cumsum() + 100,
    'volume': np.random.randint(1000, 10000, 100)
}
df = pd.DataFrame(data)

# 1. Save as CSV
df.to_csv('stock_data.csv', index=False)

# 2. Save as Pickle
df.to_pickle('stock_data.pkl')
```

DATA PROCESSING

RISKOPTIMA

Preprocessing Steps for Financial Data

1. Handling Missing Values
   • Financial data often contains missing values due to non-trading days, data collection issues, or corporate actions.
   • Options include forward/backward filling, interpolation, or removing affected rows.

2. Adjusting for Corporate Actions
   • Stock splits, dividends, and other corporate actions can create discontinuities in price series.
   • Adjusted prices account for these events to maintain consistency.

3. Normalization and Scaling
   • Different assets have different price ranges.
   • Normalization techniques like min–max scaling or standardization help models focus on relative movements rather than absolute values.

4. Handling Outliers
   • Market crashes, flash crashes, or data errors can create extreme values.
   • Techniques like winsorization, trimming, or robust scaling can mitigate their impact on models.

5. Time Series Alignment
   • When working with multiple data sources, proper time alignment is crucial.
   • Different exchanges, time zones, and reporting frequencies must be harmonized.

```python
# Data Preprocessing Example
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler,
RobustScaler

# Load stock data
df = pd.read_csv('AAPL_daily_data.csv')
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)

# 1. Handle missing values
# Check for missing values
print(f"Missing values: {df.isnull().sum()}")

# Fill missing values using forward fill
df.fillna(method='ffill', inplace=True)

# 2. Calculate returns instead of using prices
df['Returns'] = df['Adj Close'].pct_change()
```

Feature Engineering

Technical Indicators
📈 Moving Averages (MA)
📊 Bollinger Bands
🔄 Relative Strength Index (RSI)
📉 MACD (Moving Average Convergence Divergence)

Time Series Features
📅 Day, Week, Month Features
⏱️ Lag Features
💹 Return Calculations

Feature Selection and Importance
🎯 Feature Importance
🔍 Correlation Analysis
🧮 Dimensionality Reduction (PCA)

```python
# Calculate Technical Indicators
import pandas as pd
import numpy as np

def calculate_features(df):
    # Moving Averages
    df['MA5'] = df['close'].rolling(window=5).mean()
    df['MA20'] = df['close'].rolling(window=20).mean()

    # Bollinger Bands
    df['MA20_std'] = df['close'].rolling(window=20).std()
    df['upper_band'] = df['MA20'] + (df['MA20_std'] * 2)
    df['lower_band'] = df['MA20'] - (df['MA20_std'] * 2)

    # RSI calculation
    delta = df['close'].diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
    rs = gain / loss
    df['RSI'] = 100 - (100 / (1 + rs))

    return df
```

Machine Learning Models

Classification Models
🌳 Decision Trees
🌲 Random Forest
🔄 Support Vector Machines (SVM)
🧠 Artificial Neural Networks

Regression Models
📈 Linear Regression
🌿 Gradient Boosting Regression

Ensemble Models
🗳️ Voting Classifier
📚 Stacking
🔄 Bagging
🚀 Boosting (XGBoost, LightGBM)

Model Evaluation
✅ Accuracy
📊 Precision and Recall
📉 F1 Score
📈 ROC Curve and AUC

```python
# Decision Tree Model
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(
    features, target, test_size=0.2, random_state=42)

# Create and train decision tree model
dt_model = DecisionTreeClassifier(max_depth=5)
dt_model.fit(X_train, y_train)

# Make predictions
y_pred = dt_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model accuracy: {accuracy:.4f}")

# Evaluate model
report = classification_report(y_test, y_pred)
print(report)
```

Option Trading

Options Strategies

- ⬆️ Call Spread Strategy
- ⬇️ Put Spread Strategy
- ↕️ Straddle Strategy
- ↔️ Strangle Strategy
- 🦋 Butterfly Spread

ML for Options Trading

- 📊 Implied Volatility Forecasting
- 💰 Options Pricing
- 🎯 Strategy Selection
- ⏱️ Optimal Entry/Exit Timing
- 🛡️ Risk Management

```python
# Call Spread Strategy Setup
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

# Load options data
options_data = pd.read_csv('data_modules/spx_eom_options_2022.csv')

# Select options for Call Spread strategy
def setup_call_spread(options_df, date, underlying_price):
    # Filter options for specific date
    day_options = options_df[options_df['date'] == date]

    # Select ATM call option (lower leg)
    atm_call = day_options[
        (day_options['option_type'] == 'call') &
        (day_options['strike'] >= underlying_price)
    ].iloc[0]

    # Select OTM call option (upper leg)
    otm_call = day_options[
        (day_options['option_type'] == 'call') &
        (day_options['strike'] > atm_call['strike'])
    ].iloc[0]

    # Create call spread strategy
    call_spread = {
        'date': date,
        'lower_leg': atm_call['strike'],
        'upper_leg': otm_call['strike'],
        'net_premium': atm_call['mid_price'] - otm_call['mid_price']
    }

    return call_spread
```

RISKOPTIMA

Backtesting

Backtesting Components

📊 Historical Data
🔄 Signal Generation
💹 Trade Simulation
📈 Performance Analysis

Performance Metrics

💰 Total and Annualized Returns
📉 Maximum Drawdown
⚖️ Sharpe and Sortino Ratios
🎯 Win Rate and Profit Factor

Backtesting Challenges

⚠️ Overfitting Risk
💸 Transaction Costs
⏰ Look-Ahead Bias
🔄 Survivorship Bias
📏 Out-of-Sample Testing

```python
# Backtest a Strategy
import pandas as pd
import numpy as np

def backtest_strategy(df, signals, initial_capital=10000):
    # Create a positions dataframe
    positions = signals.copy()
    positions['position'] = 0
    positions.loc[signals['signal'] == 1, 'position'] = 1
    positions.loc[signals['signal'] == -1, 'position'] = -1

    # Calculate daily returns
    df['returns'] = df['close'].pct_change()

    # Calculate strategy returns
    positions['strategy_returns'] = positions['position'].shift(1) * df['returns']

    # Calculate cumulative returns
    positions['cumulative_returns'] = (1 + positions['strategy_returns']).cumprod()

    # Calculate equity curve
    positions['equity_curve'] = initial_capital * positions['cumulative_returns']

    # Calculate performance metrics
    total_return = positions['cumulative_returns'].iloc[-1] - 1
    sharpe_ratio = positions['strategy_returns'].mean() / positions['strategy_returns'].std() * np.sqrt(252)

    return positions, total_return, sharpe_ratio
```

RISKOPTIMA

RISKOPTIMA

Thank you for reading