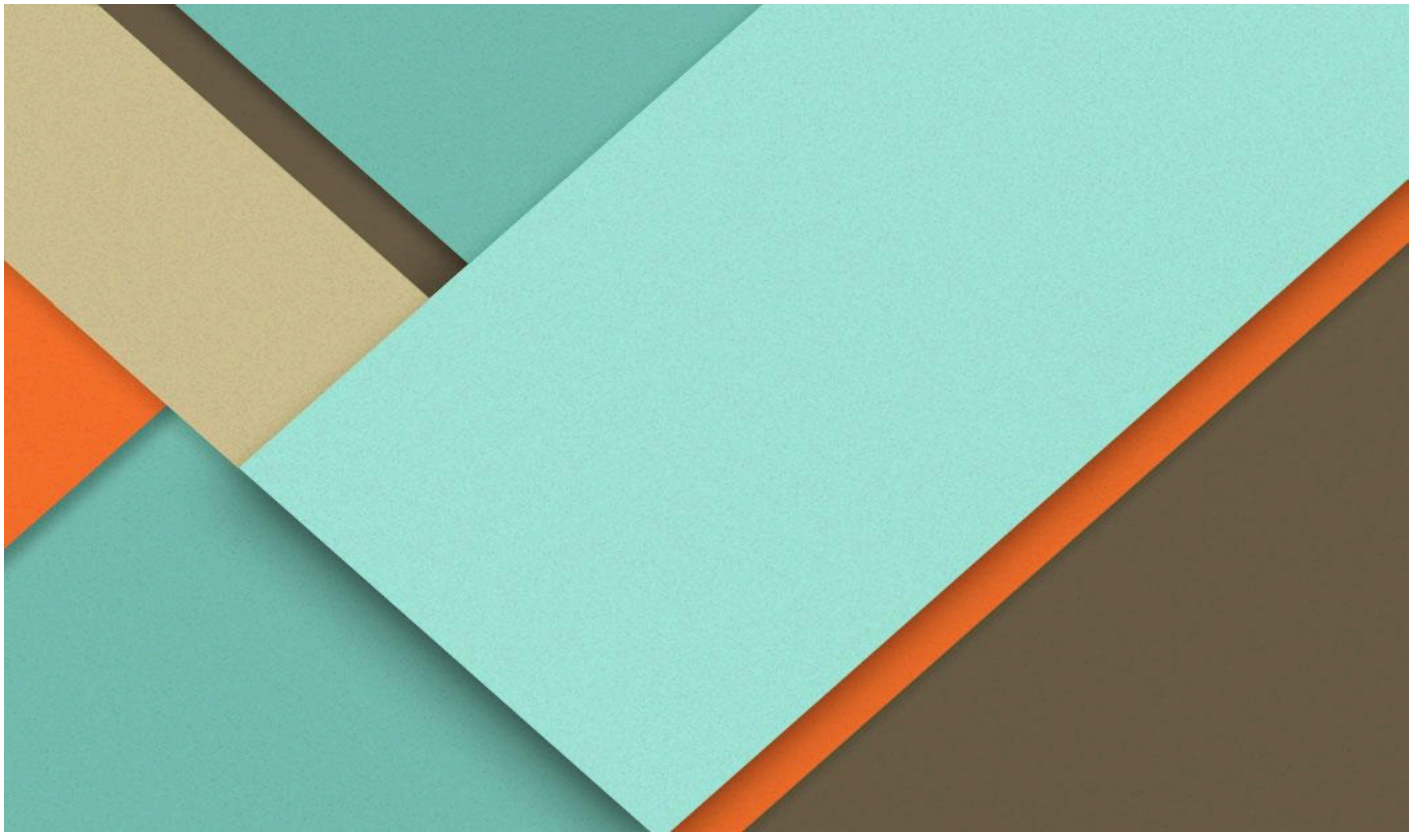


Stock Sentiment Analysis Using Machine Learning Techniques

From 25 May 2024 To 20 June 2024

—

Chaman Singhal
Enrollment -22118019
Branch - Metallurgy
IIT Roorkee

Introduction

The project aims to develop a sentiment analysis model to predict stock price movements based on newsheadlines and simulate trading strategies based on these sentiments. The goal is to determine whether sentiment analysis can provide actionable trading signals to generate profitable trades. The report outlines the process of data collection, sentiment analysis, trading signal generation, trade simulation, and performance evaluation.

Data Collection

1.1 News Headlines

I have scraped stock news headlines from the Business Insider website for a 10 specific stock ticker like Amazon,Google,Spotify etc . The scraped data includes the publication date and time, the news source, and the headline title. This information is stored in a DataFrame for further analysis.Here I am using requests and the BeautifulSoup library to scrape the web-pages.

It loops through multiple pages to gather a comprehensive dataset, which can later be used for sentiment analysis and predicting stock price movements. The code can be easily adapted for other stocks by changing the URL template and ticker symbol.

```
columns = ['datetime','ticker','source', 'title' ]
df = pd.DataFrame(columns=columns)
counter = 0
for page in range(1,200):
    url = f'https://markets.businessinsider.com/news/jpm-stock?p={page}'
    response = requests.get(url)
    html = response.text
    soup = BeautifulSoup (html, 'lxml')
    articles = soup.find_all('div', class_ = 'latest-news__story')
    for article in articles:
        datetime = article.find('time', class_ =
'latest-news__date').get('datetime')
        title = article.find('a', class_ = 'news-link').text
        source = article.find('span', class_ = 'latest-news__source').text
    #   link = article.find('a', class_ = 'news-link').get('href')
        ticker = 'JPM'       # 'AMZN', 'TSLA', 'GOOGL', 'ADBE', 'AXP', 'META', 
'NVDA' ,'SPOT' ,'MSFT' ,'JPM'
        df = pd.concat([pd.DataFrame([[datetime,ticker, source,title]], 
columns=df.columns), df], ignore_index=True)
        counter += 1

print (f'{counter} headlines scraped from {page+1} pages')
```

2.2 Stock Price Data

This line of code processes the combined DataFrame combined_stock_news by grouping the data based on the 'date' and 'ticker' columns. It then concatenates all news titles for each group into a single string. This is useful when you want to aggregate news articles by date and stock ticker, creating a summary of all news headlines for each date and stock.

```
combined_stock_news = 
combined_stock_news.groupby(['date','ticker'])['title'].apply('
'.join).reset_index()
```

This fetches historical stock price data using the yfinance library and integrates this data with the previously aggregated stock news DataFrame. The goal is to enhance the stock news data with corresponding stock price information (open, high, low, and close prices) for each date and ticker.

```
def fetch_stock_prices(ticker, date):
    stock = yf.Ticker(ticker)
    start_date = pd.to_datetime(date)
    end_date = start_date + pd.Timedelta(days=1)

    hist = stock.history(start=start_date, end=end_date)

    if not hist.empty:
        open_price = hist['Open'][0]
        high_price = hist['High'][0]
        low_price = hist['Low'][0]
        close_price = hist['Close'][0]
        return open_price, high_price, low_price, close_price
    else:
        return None, None, None, None

price_data = combined_stock_news.apply(lambda row:
fetch_stock_prices(row['ticker'], row['date']), axis=1)

combined_stock_news[['open', 'high', 'low', 'close']] =
pd.DataFrame(price_data.tolist(), index=combined_stock_news.index)
```

Calculating Price Movement

The future closing price is calculated by shifting the closing prices by the time horizon. This means for each row, the future closing price is the closing price 1 days ahead.The price movement is calculated as the percentage change between the future closing price and the current closing price.

```
time_horizon = 1

# Calculate the future closing price
stock_news['future_close'] =
stock_news.groupby('ticker')['close'].shift(-time_horizon)

# Calculate the price movement as a percentage
stock_news['price_movement'] =
((stock_news['future_close']-stock_news['close']) / stock_news['close']) * 100

# Drop rows where future_close is NaN (i.e., where we don't have enough data to
calculate the movement)
stock_news = stock_news.dropna(subset=['future_close'])
```

By creating a `movement_label` column, it prepares the data for supervised learning tasks, such as training machine learning models to predict future stock price movements based on historical data and news articles.

```
increase_threshold = 0.75
decrease_threshold = -0.75
# 1->increase, 0->no change, -1->decrease

def label_movement(movement):
    if movement >= increase_threshold:
        return 1
    elif movement <= decrease_threshold:
        return -1
    else:
        return 0

stock_news['movement_label']= stock_news['price_movement'].apply(label_movement)
```

Preprocessing

This function preprocess_text that preprocesses text data by performing several text normalization steps, such as converting to lowercase, removing non-alphabetic characters, tokenizing, removing stopwords, and stemming. The preprocessed text is then applied to the title column of the stock_news DataFrame, creating a new column preprocessed_text. This preprocessing step is crucial for preparing the textual data machine learning tasks.

```
def preprocess_text(text):

        text = text.lower()

        text = re.sub(r'[^a-z\s]', '', text)
        text = re.sub(r'[^\w\s]', '', text)

        tokens = word_tokenize(text)

        stop_words = set(stopwords.words('english'))
        tokens = [word for word in tokens if word not in stop_words]

        stemmer = PorterStemmer()
        tokens = [stemmer.stem(word) for word in tokens]

#          lemmatizer = WordNetLemmatizer()
#          tokens = [lemmatizer.lemmatize(word) for word in tokens]

        return ' '.join(tokens)

stock_news['preprocessed_text'] = stock_news['title'].apply(preprocess_text)
```

Sentiment Analysis

Sentiment Analysis for the news headlines falls under the domain of Natural Language Processing (NLP). Several methods can be employed, such as converting text to embeddings using Word2vec, TF-IDF, One-Hot Encoding, etc. These embeddings can then be used to train a supervised model based on the embedding and price movement of the stock. There are different method for sentiment analysis some are below :

4.1 FinBert pre-trained Model

Let’s utilize the FinBERT model from Hugging Face's transformers library to predict the sentiment of preprocessed text data. FinBERT is a pre-trained model specifically designed for financial sentiment analysis. Load the tokenizer and the FinBERT model from the Hugging Face model hub .The tokenizer converts text into tokens that the model can understand.

- This loop processes each text, converts it into a suitable format for the model, makes predictions, and extracts the most probable sentiment and its probability.

```
def get_finbert_sentiment(text):
    tokenizer = AutoTokenizer.from_pretrained("ProsusAI/finbert")
    tokenizer_kwargs = {"padding": True, "truncation": True, "max_length": 513}

    tokenized_text = tokenizer(text, **tokenizer_kwargs)
    sentiment_analysis = pipeline("sentiment-analysis",
model="ProsusAI/finbert" ,tokenizer=tokenizer)
    sentiment_result = sentiment_analysis(text)
    # Extract sentiment label and score
    if sentiment_result:
        sentiment_label = sentiment_result[0]['label']
        sentiment_score = sentiment_result[0]['score']

        # Convert sentiment label to numeric value
        if sentiment_label == 'negative':
            return -1 * sentiment_score
        elif sentiment_label == 'positive':
            return sentiment_score
        else:
            return 0
    else:
        return 0

# Example usage with pandas DataFrame 'data'
stock_news['finbert_sentiment_score'] =
stock_news['preprocessed_text'].apply(get_finbert_sentiment)

# Drop the intermediate column used for sentiment analysis
data = data.drop(['finbert_sentiment'], axis=1)
```

4.2 Vader Sentiment Library

VADER is particularly suitable for analyzing social media text, but it can also be used for other types of textual data. This calculates the sentiment scores for a given text and returns the negative, neutral, positive, and compound scores and these Extracted individual sentiment scores (negative, neutral, positive, and compound) from vader_sentiments are assigned to them in new columns in the DataFrame.

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
```

```
def apply_vader_sentiment(df, text_column):

    analyzer = SentimentIntensityAnalyzer()

    def get_vader_sentiment(text):
        scores = analyzer.polarity_scores(text)
        return scores['neg'], scores['neu'], scores['pos'], scores['compound']

    vader_sentiments = df[text_column].apply(get_vader_sentiment)
    df['vader_neg'] = vader_sentiments.apply(lambda x: x[0])
    df['vader_neu'] = vader_sentiments.apply(lambda x: x[1])
    df['vader_pos'] = vader_sentiments.apply(lambda x: x[2])
    df['vader_compound'] = vader_sentiments.apply(lambda x: x[3])

    # Generate Trading Signal
    def classify_sentiment(compound_score):
        if compound_score >= 0.1:
            return 1
        elif compound_score <= -0.1:
            return -1
        else return 0;

    df['vader_trading_signal'] = df['vader_compound'].apply(classify_sentiment)
    return df

stock_pre_processed_news = apply_vader_sentiment(stock_news,
'preprocessed_text')
```

Text Feature Extraction using TF-IDF

TfidfVectorizer is a method that transforms text data into numerical features based on the Term Frequency-Inverse Document Frequency (TF-IDF) statistic. This Converts textual data into numerical features using n-grams and TF-IDF statistics which is suitable for machine learning algorithms.

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from textblob import TextBlob

 # TF-IDF

tf_idf = TfidfVectorizer(ngram_range = (1,4),max_features= 10000,binary=True,
                         smooth_idf=False)

X_total = tf_idf.fit_transform(stock_pre_processed_news['preprocessed_text'])
y =stock_pre_processed_news['movement_label']
X_train = tf_idf.fit_transform(X_train)
X_test = tf_idf.transform(X_test)
```

Trading Signals

Sentiment scores are aggregated by date, and trading signals are generated based on the sentiment scores. The sentiment score threshold can be tuned accordingly to obtain a better result.

We can define a threshold above which we consider the sentiment positive (suggesting a potential increase in stock price) and below which we consider it negative (suggesting a potential decrease in stock price) and neutral if between them.

```
def generate_trading_signal(sentiment_score):
      if sentiment_score > buy_threshold:
            return 1
      elif sentiment_score < sell_threshold:
            return -1
      else:
           return 0

data['trading_signal'] =
data['finbert_sentiment_score'].apply(generate_trading_signal)
```

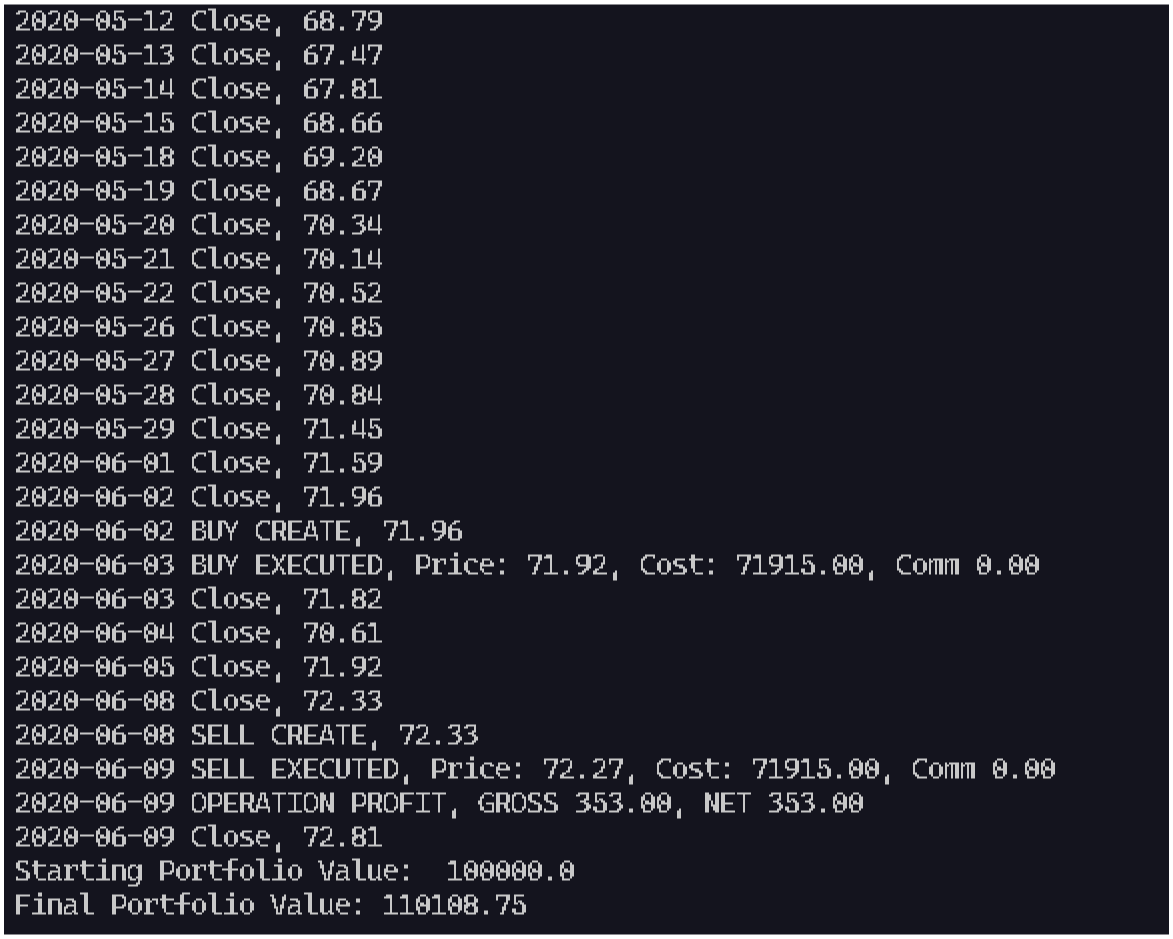
Trading Simulations using BackTrader Library

I have used the **Backtrader library**, where trading signals are generated based on sentiment scores and the strategy follows a basic buy-and-hold approach with a fixed exit condition. This strategy logs various actions and decisions to the console for tracking purposes.

At the core of the strategy, it references the closing prices (`dataclose`) and sentiment scores (`datasentiment`) from the provided data. The strategy initiates by logging the closing prices and checking for any pending orders to avoid placing multiple orders simultaneously. The primary logic is to enter the market (buy) when the sentiment score exceeds 0.6. When this condition is met, it creates a buy order for 1000 units and logs the transaction details.

```
class SentimentStrat(bt.Strategy):
    params = (
        ('exitbars', 3),
    )

    def log(self, txt, dt=None):
        dt = dt or self.datas[0].datetime.date(0)
        print(f'{dt.isoformat()} {txt}') # Print date and close

    def __init__(self):

        self.dataclose = self.datas[0].close
        self.datasentiment = self.datas[0].sentiment

        # To keep track of pending orders and buy price/commision
        self.order = None
        self.buyprice = None
        self.buycomm = None
    def notify_order(self, order):
        if order.status in [order.Submitted, order.Accepted]:
            return

        # Check if an order has been completed
        # Attention: broker could reject order if not enough cash
        if order.status in [order.Completed]:
            if order.isbuy():
                self.log(
                   'BUY EXECUTED, Price: %.2f, Cost: %.2f, Comm %.2f' %
                    (order.executed.price,
                        order.executed.value,
                        order.executed.comm)
                )
                self.buyprice = order.executed.price
                self.buycomm = order.executed.comm
            else: # Sell
                self.log(
                  'SELL EXECUTED, Price: %.2f, Cost: %.2f, Comm %.2f' %
                    (order.executed.price,
                        order.executed.value,
                        order.executed.comm)
                )

            self.bar_executed = len(self)

        elif order.status in [order.Canceled, order.Margin,
order.Rejected]:
            self.log('Order Canceled/Margin/Rejected')

        self.order = None

    def notify_trade(self, trade):
        if not trade.isclosed:
            return

        self.log('OPERATION PROFIT, GROSS %.2f, NET %.2f' %
                    (trade.pnl, trade.pnlcomm))

    def next(self):
        # Simply log the closing price of the series from the reference
        self.log('Close, %.2f' % self.dataclose[0])

        # Check if an order is pending ... if yes we cannot send a 2nd
one
        if self.order:
            return

        # Check if we are in the market
        if not self.position:

            # If the sentiment score is over 0.6, we buy
            if self.datasentiment[0] > 0.6:

                self.log('BUY CREATE, %.2f' % self.dataclose[0])

                # Keep track of the created order to avoid a 2nd order
                self.order = self.buy(size=1000)

        else:
            # Already in the market, we sell three days (bars) after
buying:
            if len(self) >= (self.bar_executed + self.params.exitbars):

                self.log('SELL CREATE, %.2f' % self.dataclose[0])
                # Keep track of the created order to avoid a 2nd order
                self.order = self.sell(size=1000)
```

Once a buy order is executed, the strategy monitors the holding period. It sells the asset exactly three days (bars) after the buy order was executed, regardless of the sentiment score at that time. It ensures that only one order is active at any time to prevent conflicting trades. The strategy also includes logging for order status updates, such as order completion, cancellation, or rejection, and it calculates and logs the profit when a trade is closed.



```
2020-05-12 Close, 68.79
2020-05-13 Close, 67.47
2020-05-14 Close, 67.81
2020-05-15 Close, 68.66
2020-05-18 Close, 69.20
2020-05-19 Close, 68.67
2020-05-20 Close, 70.34
2020-05-21 Close, 70.14
2020-05-22 Close, 70.52
2020-05-26 Close, 70.85
2020-05-27 Close, 70.89
2020-05-28 Close, 70.84
2020-05-29 Close, 71.45
2020-06-01 Close, 71.59
2020-06-02 Close, 71.96
2020-06-02 BUY CREATE, 71.96
2020-06-03 BUY EXECUTED, Price: 71.92, Cost: 71915.00, Comm 0.00
2020-06-03 Close, 71.82
2020-06-04 Close, 70.61
2020-06-05 Close, 71.92
2020-06-08 Close, 72.33
2020-06-08 SELL CREATE, 72.33
2020-06-09 SELL EXECUTED, Price: 72.27, Cost: 71915.00, Comm 0.00
2020-06-09 OPERATION PROFIT, GROSS 353.00, NET 353.00
2020-06-09 Close, 72.81
Starting Portfolio Value:  100000.0
Final Portfolio Value: 110108.75
```

Portfolio for Google stocks with initial capital of 100000 dollar

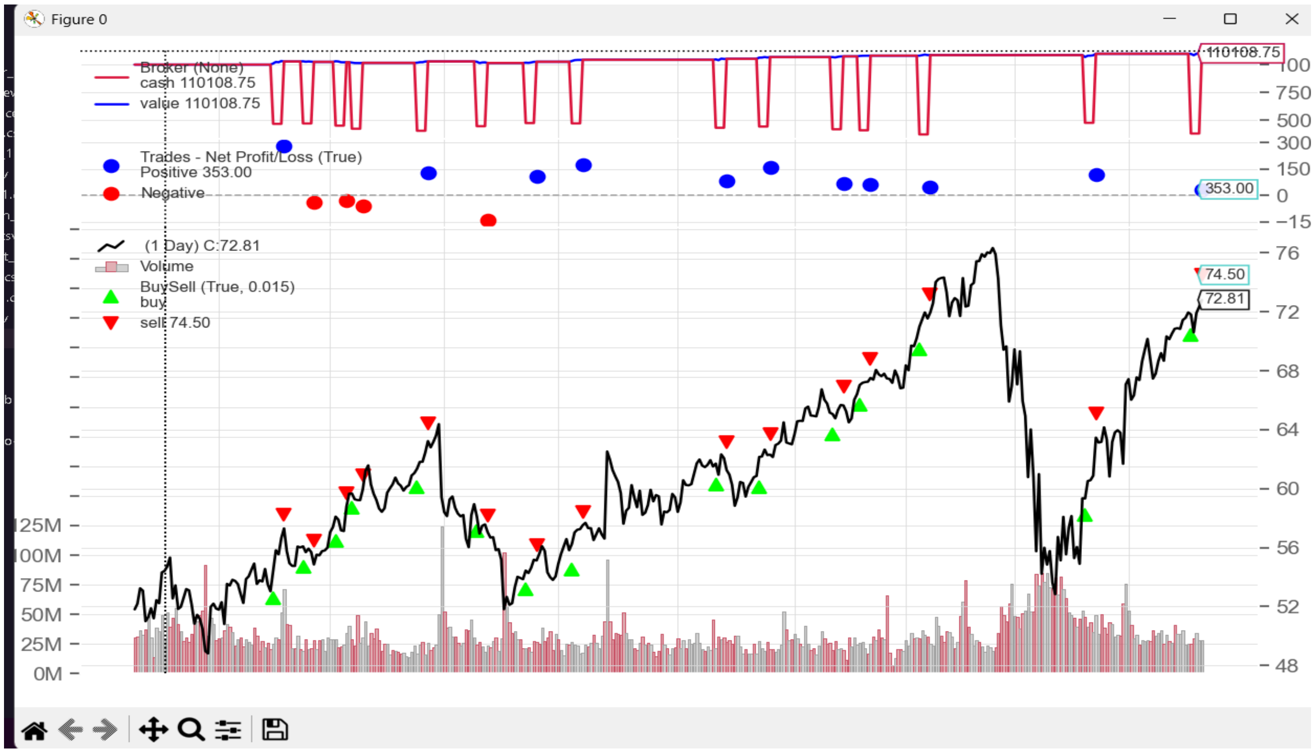
Visualization

Backtesting is a method of testing your trading strategy on historical market data, to evaluate how your strategy would have performed in the past. Obviously, past performance does not guarantee any future performance. In fact, the stock market is known for being volatile, dynamic, and nonlinear. While backtesting a strategy does not predict exactly how it will perform in the future, it can help assess the utility of a strategy.

```
# Instantiate the Cerebro engine
cerebro = bt.Cerebro()
# Add the strategy to Cerebro
cerebro.addstrategy(SentimentStrat)
```

```python
# Add the data feed to cerebro
cerebro.adddata(data)
# Add an analyzer to get the return data
cerebro.addanalyzer(bt.analyzers.PyFolio, _name='PyFolio')

# set initial porfolio value at 100,000$
cerebro.broker.setcash(100000.0)
start_value = cerebro.broker.getvalue() # should be 100,000$

# Run the Backtest
results = cerebro.run()

# Print out the final result
print('Starting Portfolio Value: ', start_value)
print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())

# Plot the results using cerebro's plotting facilities
cerebro.plot()
```

Now, use the *Cerebro engine* plotting function to have a more detailed feedback over the backtest:



Performance Metrics

1.1 Portfolio Metrics

The number of trades, win percentage, and total profit are calculated to evaluate the performance of the trading strategy.

1.2 Sharpe Ratio and Maximum Drawdown

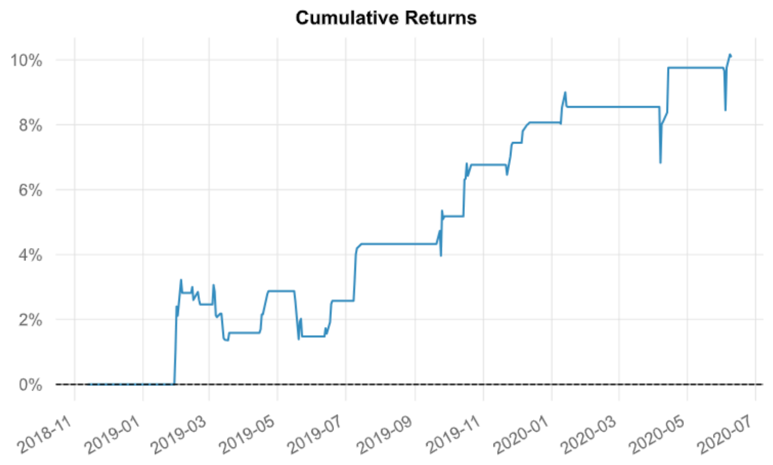
The Sharpe Ratio and Maximum Drawdown are calculated to provide additional insights into the risk-adjusted performance and downside risk of the trading strategy.

I have calculated all these matrices with the help of a library named quantstats to obtain a more detailed feedback:

```
strat = results[0]

# Uses PyFolio to obtain the returns from the Backtest
portfolio_stats = strat.analyzers.getbyname('PyFolio')
returns, positions, transactions, gross_lev =
portfolio_stats.get_pf_items()
returns.index = returns.index.tz_convert(None)

import webbrowser
# Feeds the returns to Quantstats to obtain a more complete report over
the Backtest
quantstats.reports.html(returns, output='stats.html', title='Google
Sentiment')
webbrowser.open('stats.html')
```

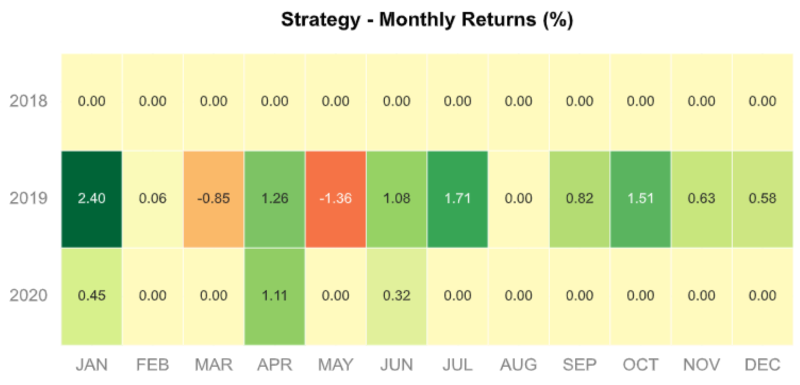
Google Sentiment 13 Nov, 2018 - 9 Jun, 2020

Generated by QuantStats (v. 0.0.62)



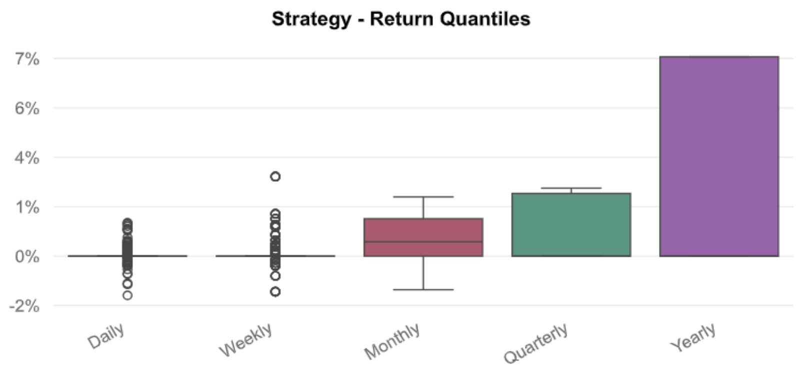
Key Performance Metrics

Metric	Strategy
Risk-Free Rate	0.0%
Time in Market	20.0%
Cumulative Return	10.11%
CAGR﹪	4.32%
Sharpe	1.61
Prob. Sharpe Ratio	98.16%
Smart Sharpe	1.41
Sortino	2.81
Smart Sortino	2.46
Sortino/√2	1.99
Smart Sortino/√2	1.74
Omega	1.95
Max Drawdown	-2.0%



Strategy - Monthly Returns (%)

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
2018	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2019	2.40	0.06	-0.85	1.26	-1.36	1.08	1.71	0.00	0.82	1.51	0.63	0.58
2020	0.45	0.00	0.00	1.11	0.00	0.32	0.00	0.00	0.00	0.00	0.00	0.00



Conclusion

The project demonstrates that sentiment analysis of news headlines can be used to generate trading signals. A trading algorithm thus can be designed to respond to the trading signals generated by the sentiment analysis. Thus in this project I have demonstrated the use of machine learning methods applied in the financial domain for the buying and selling of stocks and portfolio management. The use of NLP techniques for the sentiment analysis of the news headlines, thus generating trading signals is successfully implemented in the code snippets given above

References

- https://medium.com/@chedy.smaoui/using-sentiment-analysis-as-a-trading-signal-beat-the-market-with-transformers-54c65641bec8
- https://blog.quantinsti.com/sentiment-analysis-trading/
- https://markets.businessinsider.com/