Assignment #2: Sentiment Analysis

Total score: 130
This assignment can be completed individually or by a team of a maximum of three members.
Due date: See the course page

Objectives

Build sentiment analysis models and apply them to financial news to estimate the impact of news on stock prices.

Datasets

Two financial news datasets collected for several years (2009-2020) by publishers are provided:

- "analyst_ratings.csv" – Headlines collected by Publisher A, which may include the price targets estimated by professional analysts through complex modeling to estimate the current intrinsic value of the company.
- "headlines.csv" – Headlines collected by Publisher B, without price target information.

Both datasets follow a similar schema:

analyst_ratings(*id, headline, URL, publisher, date, symbol*)

- id: row number
- headline: headline news that may include the target price (not all headlines indicate the target price).
- URL: the URL of the headline news
- Publisher: news publisher
- Date: published date and time
- stock: stock symbol representing the company used by traders in the market.

headlines*(id, headline, URL, publisher, date, symbol)*

Problem Context

Stock price fluctuates daily due to company-specific developments, sector-wide trends, market sentiments, and/or unpredictable external factors. Since a company's **intrinsic value** is unknown, predicting stock price movements is extremely challenging. In this assignment, you will develop sentiment analysis models that can estimate the potential impact of financial news on future stock prices and write a brief analysis report about the modeling results.

Organization of Program Files and Directories

The required program files and directory structure are provided in "Assignment_2.zip" for efficient grading. Before getting started, review the helper functions and test file: "util.py", "webscraping.py", "webscraping_headless_parsing.py", "dataset_schema.py", "test_dataset_schema.py". These files are provided to help you standardize and validate your CSV files. Each file in the directories is currently blank and must be completed by you. The file names correspond to the required tasks described in the sections below. **Do not change** any file names, directory names, or the directory structure. For example, the program "2_trading_sim_eval.py" should be executable with the command: "**python** 2_trading_sim_eval.py" without requiring additional parameters or instructions.

Required Tasks

Phase 1: Data Collection and Preprocessing

1.1 Collect historical prices, S&P 500 indices, and news data

Complete the program "**1_collect_data.py**" for the following task.

(a) Download historical daily prices for each stock symbol in the datasets, along with S&P 500 index for the same period, from a freely available data sources (e.g., Yahoo Finance) using any available Python package (e.g., "yfinance" or "yahoo-fin"). Save the price data locally in a file named "historical_prices.csv". For each symbol, the required fields are at least: "symbol", "date", "open", "high", "low", "AdjClose", and "volume". Note that adjusted close provides a more accurate price than the raw close price. So, we will use it for close price. For consistency, treat the daily S&P 500 index values like stock prices with the symbol "s&p", and include them in historical_prices.csv. You may include additional fields if needed for more detailed analysis. The schema for historical_prices.csv will be:

historical_prices*(date, symbol, open, high, low, close, volume)*

(b) Download the full news articles from the URLs provided in the dataset for **at least six consecutive years**. These articles will be used to create training and test datasets. You

may use Python web scrapping tools (e.g., requests, BeautifulSoup, playwright, selenium, etc.).

Note: When scraping, please follow ethical and legal guidelines

- Check each website's robots.txt file to ensure scraping is allowed.
- Respect rate limits to avoid overloading servers.
- Use official APIs where available, since they may provide more reliable and compliant access to the data.

Full new articles can be downloaded from several sites, including Benzinga, Zacks, Gurufocus, and Seekingalpha. However, Gurufocus appears to block any attempts at automated scrapping. Refer to the sample code for web scraping.

If none of the sites allowed scraping, this step will be skipped.

(c) Create a **new dataset** "**all_news.csv**" by merging the two datasets. Include all fields from the original datasets, removing redundant content if any, and add a new field "article" for the full news articles. The schema for "all_news.csv" will be:
all_news*(date, symbol, headline, URL, article, publisher)*

1.2 Calculate volatility

Complete the program "**2_calculate_volatility.py**" for the following task.

We assume that impactful news affects the stock price for about three days. The average daily volatility (implied daily risk) for each company and the S&P 500 index is calculated as the standard deviation of returns over three trading days, expressed as a percentage.

- **Daily log return**

  From the simple return in percentage $R_t = \frac{P_t - P_{t-1}}{P_{t-1}} = \frac{P_t}{P_{t-1}} - 1$ and continuous compounding formula $P_t = P_{t-1}e^{r_t}$, we can calculate the daily log returns:

  $r_t = \ln(\frac{P_t}{P_{t-1}})$ or $r_t = \ln(1 + R_t)$ where P_t is the price at time t and P_{t-1} is the price at previous time (three trading days ago).

- **Daily volatility**

  The standard deviation of returns provides the average daily volatility: σ_w= stdev(r_t) for each stock and S&P 500 index. Optionally, periodic volatility can be calculated for longer intervals: $\sigma_{period} = \sigma_w \times \sqrt{time}$. For example, the annualized volatility: $\sigma_{annual} = \sigma_w \times \sqrt{252}$ (assuming 252 trading days in a year). Note: Volatility is the square root of variance and can be interpreted as risk. Risks accumulate additively in variance, so annual volatility can be considered as yearly risk.

- **Market-adjusted return and volatility**

  Stock prices are influenced by both market sentiment (reflected in S&P 500 volatility) and company-specific news. To account for market influence, we can calculate the market-adjusted return (also called abnormal return) and market-adjusted volatility by **regressing** the stock's daily log returns on the market's daily log returns.

  The **market model** is defined as: $\boldsymbol{r_{a,t} = \alpha + \beta r_{m,t} + \epsilon_t}$

  where:

  - $r_{a,t}$ = market model, observed asset a return at time t (that is r_t)
  - $r_{m,t}$ = market (S&P 500) log return at time t
  - $\hat{\alpha} = \bar{r_t} - \hat{\beta}\bar{r_{m,t}}$ (Intercept, expected return after accounting for market exposure)
  - $\hat{\beta} = \frac{Cov(r_t, r_{m,t})}{Var(r_{m,t})} = \frac{\sum_t^N (r_t - \bar{r_t})(r_{m,t} - \bar{r_{m,t}})}{\sum_t^N (r_{m,t} - \bar{r_{m,t}})^2}$ (stock's sensitivity to the market)
  - ϵ_t = idiosyncratic return (residual)

  The **market-adjusted return** (or idiosyncratic return after removing the market effect) is: $\boldsymbol{\hat{\epsilon_t} = r_{a,t} - (\hat{\alpha} + \hat{\beta} r_{m,t})}$

  The **market-adjusted volatility** (or idiosyncratic volatility) is:

  $$\boldsymbol{\sigma_{\varepsilon_t} = \text{stdev}(\varepsilon_t) = \sqrt{\frac{1}{N-1}\sum_t^N (\varepsilon_t - \bar{\varepsilon})^2}}$$

  The **asset volatility** (combining market-driven and idiosyncratic components) is: $\boldsymbol{\sigma_{r_{a,t}} = \text{stdev}(r_{a,t})}$ where $\boldsymbol{r_{a,t} = \alpha + \beta r_{m,t} + \epsilon_t}$

- **Interpretation**

  Beta ($\boldsymbol{\beta}$) measures how sensitive your stock is to the market, capturing the systemic risk (the part of stock volatility explained by market movements. So, $\beta = 1$, stock moves in line with the market; $\beta > 1$, the stock is more volatile than the market; $\beta < 1$, stock is less volatile than the market; $\beta < 0$, the stock moves opposite to the market.

  Alpha ($\boldsymbol{\alpha}$) represents abnormal/excess return, that is, the expected return of the stock after accounting for its exposure to the market. So, $\alpha > 0$, stock outperforms what's expected given its beta; $\alpha < 0$, stock underperforms what's expected given its beta.

  Residual ($\boldsymbol{\epsilon_t}$) is the idiosyncratic return, the portion of the stock return not explained by the market.

1.3 Estimate impact scores

Complete the program "**3_estimate_impact_scores.py**" for the following task.

(a) Calculate the impact score based on the **return** and **volatility**, and map it into a discrete range [-3, +3] as impact scores, where -1, -2, -3 = negative impact, 1, 2, 3 = positive impact, and 0 = neutral, formally: $impact\_score_t = sign(r_t) \cdot f(\sigma_t)$.

- Return $\boldsymbol{r_t}$ captures the direction of the market reaction. For computing the impact score, we will use the **market-adjusted return** $\boldsymbol{\epsilon_t}$ instead of the daily return r_t.
- Volatility $\boldsymbol{\sigma_t}$ captures the magnitude/uncertainty of the market reaction. We will use the **total stock volatility** $\boldsymbol{\sigma_{r_{a,t}}}$ (including both market-driven and idiosyncratic components) instead of daily volatility σ_t.
- **Implication of the impact score**: A large impact score indicates a large positive return with high volatility (+3), a small impact score indicates a small positive return with low volatility (+1), a large negative impact score indicates a large negative return with high volatility (-3), and other combinations follow similarly.

Z-score normalization and composite score

- Normalize both return and volatility using z-scores:

Normalized return $z_r = \frac{\epsilon_t - \mu_r}{\sigma_r}$ and volatility $z_\sigma = \frac{\sigma_{\varepsilon_t} - \mu_\sigma}{\sigma_\sigma}$

- Define a composite impact score function that maps return and volatility into impact scores:

$$\boldsymbol{impact\_score_t} = \begin{cases} \mathbf{0}, & \textbf{if } |z_r| \leq 0.5 \\ \boldsymbol{sign(z_r)} \cdot \big(\mathbf{1} + \boldsymbol{I}(|z_r| > 1) + \boldsymbol{I}(z_\sigma > 1)\big), & \textbf{if } |z_r| > 0.5 \end{cases}$$

  where:
  - $I(\cdot)$ is an indicator function that equals 1 if the condition is true, 0 otherwise.
  - Return thresholds: Large move if $|z_r| > 1$, small move if $0.5 < |z_r| \leq 1$, neutral: $|z_r| \leq 0.5$
  - Volatility thresholds: $z_\sigma > 1$, normal volatility otherwise

(b) Create a new dataset "historical_prices_impact.csv" from historical_prices.csv" by adding the returns, volatilities and impact score calculated from **sections 1.2 and 1.3**.
The schema for historical_prices_impact.csv will be:
historical_prices_impact*(date, symbol, open, high, low, close, volume, daily_return, daily_volatility, market_return, beta, alpha, market_adj_return, market_adj_volatility, impact_score)*

1.4 Identify sentiment words and vectorize the news data

Complete the program "**4_identify_and_vectorize.py**" for the following task.

(a) From the datasets, **identify** the **top 10 sentiment-bearing words** that are generally known to influence stock prices (e.g., gain, loss, strong, weak, upgrade, downgrade, etc.), **list them** with a brief justification of your choices based on the supporting evidence, and **explain how** these words were incorporated into the vectorization process.

(b) Create a new dataset "aggregated_news.csv" from "all_news.csv" by aggregating all the news (both headlines and full articles) over three consecutive trading days. This aggregation assumes that stock prices are influenced by news from the past three trading days. Specifically, we assume that the current impact score at time t reflects news published at time t as well as the previous two trading days ($t, t-1, t-2$). The schema for "aggregated_news.csv" will be:

aggregated_news*(date, symbol, news)* where *news* is aggregated news.

(c) Preprocess the news data for sentiment analysis.

The headlines or full news articles are in text or HTML format and can contain unnecessary text content. As discussed in class, the possible text preprocessing tasks are:

- **Tokenization and text normalization**: extracting tokens, lowercasing, removing punctuation, etc.
- **Relevant text extraction**: to ensure that only the main content of the news article or headline is retained, removing irrelevant contents such as HTML tags, advertisement, navigation menus, or unrelated HTML content.
- **Additional text processing**: to handle stop words, negation, occurrence of important words, etc. (see slide #29).
- Optional preprocessing steps: stemming or lemmatization, or handling special characters.

After preprocessing, briefly describe the preprocessing methods you applied and explain why they are important for sentiment analysis.

(d) Vectorize the aggregated news in "aggregated_news.csv" using a Document-Term Matrix (DTM), a TF-IDF weighted DTM, and a custom (curated) feature matrix (see slide #43 for an example) and create new datasets by adding the impact score from "historical_prices_impact.csv". The schema for each dataset will be:

vectorized_news_dtm*(date, symbol, news_vector, impact_score)*
vectorized_news_tfidf*(date, symbol, news_vector, impact_score)*
vectorized_news_curated*(date, symbol, news_vector, impact_score)*

Phase 2: Text Classification for Sentiment Analysis

2.1 Data preparation

Complete the program "**1_process.py**" for the following task.

Dataset selection and splitting for each vectorized dataset in section 1.4 (d)

- Select a consecutive time segment of at least **three years** and split it into training (**80%**) and testing (**20%**) sets, maintaining sequential order within each segment. The selected segment may come from the beginning, middle, end, or all of the dataset, but the split must **preserve sequential time order** to avoid mixing future data into the past or **prevent lookahead bias**.

2.2 Classification modeling

(a) MLP design for training

Complete the program "**2_model.py**" for the following task.

Design a MLP in Pytorch to train a classifier on **each vectorized dataset** created in **section 1.4 (d)** and clearly describe the key architectural choices, including:

- Number of layers, Number of neurons per layer, Activation functions
- Other hyperparameters (e.g., learning rate, batch size, optimizer, number of epochs).

(b) Training

Complete the program "**2_training.py**" for the following task.

Train the model and save the model as "[name of your model].pth". You can do this through torch.save(). For further information, refer to PyTorch.

(c) **Evaluation**

Complete the program "**4_eval.py**" for the following task.

Load the model you saved in step (b). Evaluate and compare the performance of each classifier in terms of **% accuracy** and discuss the results in the report file.

Phase 3: Trading Simulation

3.1 Trading rules

Complete the program "**1_trading_rules.py**" for the following task.

Assumption:

- All orders are executed at the daily closing price.
- No transaction fees are considered.

Impact score estimation and trading strategies

For each day, **estimate** the impact score for the past three trading days of news using the **trained model**. For trading decisions, use the **impact scores predicted** by the model (learned during training) rather than the precalculated impact scores from Section 1.3. Implement a simple sentiment-driven trading system based on the following strategies:

Buy rule: If news has a positive score, buy x number of shares:

$$x = \max\left[1, floor\left(\frac{\alpha \cdot s\% \cdot b}{p}\right)\right] \text{ if } \frac{b}{p} \geq 1, \text{ 0 otherwise}$$

where:

- p = stock price, s = impact score, b = current cash balance, α = a multiplier ≤ 10 (e.g., $\alpha = 2$, used to determine the amount of investment for each order)
- $\alpha \cdot s\% = \alpha \times impact\_score\%$ (e.g., if $\alpha = 2$ and $s = 2$, then $2s\% = 4\%$).

Buy only if the balance is sufficient enough to purchase at least one share. For example, the current balance b = \$20,000, stock price p = \$10.3,

and impact score s = 2, then $x = floor\left(\frac{0.04 \times 20000}{10.3}\right) = 77$. So, you buy 77 shares.

Sell rule: If news has a negative score and you own at least one share (no short selling allowed), sell y number of shares: $y = \begin{cases} r - x, & \text{if } r \geq x \\ r, & \text{if } r < x \end{cases}$.

where:

- $r > 0$ = number of shares currently owned
- x = number of shares calculated from the buy rule above

Neural rule: If new is neutral, take no action (no buy or sell).

3.2 Trading simulation for performance evaluation

Complete the program "**2_trading_sim_eval.py**" for the following task.

(a) Initial setup and liquidation

- Assume your account starts with an initial balance of **$100,000** on the first trading day in **the dataset**. All **Buy/Sell orders are assumed to be filled immediately** at the requested close price with no trading fees.
- **On the final trading day** in the dataset, **sell all remaining stocks** at the closing price, ensuring that the account balance is fully in cash.

(b) Trade tracking and performance metrics

For each trade, **calculate**:

- $gain/loss for the trade
- Total $gain/loss for all trades
- Annual % return,
- Total % return relative to the initial balance
- Available cash for trade

Log all trade data into a file "**trade_log.csv**", including the following fields:

- Transaction date
- Symbol
- Trade type (Buy/Sell)
- Number of shares
- Price
- Transaction amount
- Available cash after the trade
- News (headline)
- Impact score

Log the final summary into a file "**final_summary.csv**", including the following fields:

- Total $gain/loss
- Average annual % return
- Total % return
- Final account balance

Note: The log file serves as a valuable resource for debugging and validating your trading simulation results.

Phase 4: Analysis and Reflection

4.1 Summary and conclusion

(a) Briefly summarize your overall sentiment analysis results, highlighting key observations. Evaluate the effectiveness of trading strategies informed by the return and volatility-based impact scoring method and discuss any limitations of the method, including assumptions made, potential biases, or data constraints.

(b) Reflect on your experience, including challenges faced, insights gained, and lessons learned from implementing the workflow.

4.2 Improvement and future work

(a) Suggest any potential improvements or alternative approaches that could achieve more accurate sentiment analysis in future studies, e.g., better feature engineering, more sophisticated NLP models, alternative financial metrics, or improved market-adjustment methods, etc.

(b) Optionally, identify any logical or mathematical flaws in the impact score method and trading strategies implemented in this assignment. Discuss how these could be addressed or mitigated in future work.

Deliverables

- **One report file** (PDF or Word) that includes:
    - All team member names and % contribution of each member. If every member contributed equally, simply state "equal contribution." If there is disagreement on contributions, provide a brief task description for each member. In this case, different grades may be assigned individually.
    - All answers and results from the required tasks
    - When including example data in your report, show only a few representative samples instead of the entire dataset.

- **All Python program files** you created, and **CSV files generated** from your programs. DO not upload the original news datasets or Docker images. Please contact Tenshi (the grader) if the total size exceeds 1 GB. In that case, include only a portion of your datasets.

Grading Criteria

- Quality of work shown on the report (e.g., modeling process and results, methods used, and correctness of implementation and analysis)
- Level of understanding of related subjects as reflected in the report
- Effort (10%)