CS 410 Software Code Documentation

December 9, 2021

Team AHR

Members (Team Captain in bold)

- **Anthony Petrotte (adp12@illinois.edu)**
- Hrishikesh Deshmukh (hcd3@illinois.edu)
- Rahul Jonnalagadda (rjonna2@illinois.edu)

Overview

The overall function of our code is firstly, to analyze the financial news cycle in different time intervals to create a time interval sentiment metric on particular global securities. Secondly, we compile the intermediate sentiment results during the metric calculation into a time series dataset that can be compared to price movement in the underlying security. This dataset potentially has many uses, including the possibility to aid in identifying securities that are more prone to volatility from volume in individual (and potentially more naïve) investors trading in a more impulsive/emotional manner.

By collecting a corpus of recent news focused on a specific company/security and trimming it down into subsequently smaller relevant sub-documents, we have created a time-series sentiment calculation that can be compared to the price of the company/security. If you are familiar with the concept of a stock price indicator/oscillator, the resulting dataset can be used in a similar manner. This is a useful tool or addition to the task of stock screening, and could be implemented as an addition to a computational trading strategy.

Usage and Setup

The following documentation of the usage of software includes the documentation of usage of APIs. Python dependencies and libraries such as Hugging Face's datasets library may need to be installed to run the software. We utilized APIs to get new data and ticker information. The subsequent API keys were stored in the config.py file under their respective variables and can optionally be stored in a file named keys.py, which will be part of a user's local codebase and not committed to GitHub.

- Recommended APIs to get started
    - API to get news data
        - Polygon.io

            - Free for news (pricey for market data)
            - Best option for quickly getting started compiling news
            - Very large page size limit (1000)
            - Not good for company info, especially outside of the US
        - API to get ticker information
            - AlphaVantage
                - Free
                - Simple one-click api key to copy paste
- Optional APIs (supported)
    - NewsApi (formerly google, now NewsAPI.org)
        - Free Developer Version
        - Only past 1 month's news
        - Daily call limit: 100
        - Page size limit: 100
    - Currents API
        - Free
        - Daily call limit: 600
        - Page size limit: 200
    - Usearch (formerly contextual_web, now Web Search on RapidAPI)
        - Need Rapid API account
        - Free-mium
        - Daily call limit: 100
        - Page size limit: 50
        - Rate limit: 1 per second
    - Yahoo Finance
        - Good for quick price info
        - Was previously good for getting company info before it stopped working all of a sudden

Implementation

In the section below, we have also provided documentation with sufficient detail of how the software is implemented and how all components function together. This will allow others to have a basic understanding of our code for future extension or any further improvements.

- Setup
    - Pull in api keys from config
    - create a Ticker() object: passing in config
    - Initiate a Corpus object
- Getting web results and initiating Corpus

    - Using web_query:
        - Initiate a web_query object
            - Query_all(): Runs through the available apis and makes threaded calls to collect result urls
            - Compile_results(): combines the results of the api responses and deletes duplicate urls
            - Scrape_results(): scrapes the list of urls and compiles the raw website text to be given to the Corpus
            - Get_results(): returns the stored dataframe of urls and text
    - Setting up the Corpus:
        - Use the set_results() function to store the results from the web_query in the corpus for processing (needed for dataset building)
        - Use set_corpus() with the web_query documents and urls
- Setting up Ranker and Initial Queries
    - Initiate the ranking function (in our case, custom built BM25 object class from util.pyRanker
    - Fit the ranker to the corpus documents
    - Build the queries used by the ranking function using build_queries() and passing in the ticker object from step 0
- Initial Ranking
    - Rank corpus documents for relevance with rank_docs() passing in the ranker
    - Prune the documents with prune_docs(). This is pre-set to only prune 0 ranked documents on the primary query using a standard BM25 score
        - Note: these documents are not removed, only indexed in the pruned_docs for the sub-division process
- Sub-Dividing
    - Create dictionary of sub-docs from the original documents by calling sub_divide() and passing in the Transformer's tokenizer(if needed)
        - Note: The tokenizer is needed to ensure that the length of the subdocs created will not exceed the maximum token size used by the Transformer. The Corpus sets the standard distil-BERT tokenizer on initiation.
    - Rank the newly created subdocs using rank_subdocs() and pass in the ranker
    - Prune the subdocs with prune_subdocs(). This is pre-set similarly to prune_docs() and prunes 0 ranked subdocs using the prime query and standard BM25 score
        - Note: Like the prune_docs() function, this does not remove any subdocs, but creates an index of the ones to keep from the subdoc dictionary
- Relevant Set
    - after sub-dividing and pruning out all the trash, make the relevant set by calling make_relevant()
    - Rank the relevant set with rank_relevant().

        - □ This ranking function is pre-set to run a BM25 with Structured Query Expansion. The expanded query and its weights set can be adjusted when initiating build_queries()
    - ○ If needed/wanted, you can further prune the relevant set using prune_relevant()
        - □ Note: unlike the other two pruning functions, this directly adjusts the relevant_set and relevant_scores stored by the corpus
- Sentiments
    - ○ get_sentiments(): Once a relevant set is established
        - □ Using Transformer's classifier will run each relevant subdoc through the classifier to produce a sentiment score
- Dataset Initiation
    - ○ data_preprocess() will setup the needed dictionaries for creating pandas dataframes
        - □ This is also where the scores are calculated, which are simply the sentiment weighted by the relevance
    - ○ build the two dataframes with build_fulldf() and build_pricedf()
- Dataset for graphing
    - ○ initiate a data_manager object from util.data_manager.
        - □ the data_manager() needs the location of the '_data' directory on initiation
    - ○ tell the data_manager to put the data in a retrievable place with data_manager.store_date() and pass in the ticker symbol and dataframes
- Graphing
    - ○ get_ticker_list() calls the datamanger to return the available list of stored tickers
    - ○ get_pricedf() will tell the datamanger to return the stored price_df.csv as a pandas dataframe. This should have everything needed to graph
    - ○ Get_fulldf() will return the full_df.csv. This is all of the data needed for recalculation, or recompiling the relevant documents as it holds urls

Team Contribution

Below, we have briefly described the contributions of each team member. All members were able to contribute a fair amount of effort and time to the project.

Anthony Petrotte

- The rockstar
- Lead group by delegating tasks and responsibilities
- Co-wrote and researched project proposal
- Lead architect for system design and implementation
- Lead back end developer
- Set up APIs for web queries

- Created ranking and sub division of corpus documents
- Created data set initiation and data pipeline for graphing

Hrishikesh Deshmukh

- Co-wrote and researched project proposal
- Lead front end developer for interactive user interface
- Implemented plotly dashboard of price, sentiment score, etc.
- Completed team progress report
- Set up final relevancy calculations and relevant set
- Debugged and documented codebase

Rahul Jonnalagadda

- Co-wrote and researched project proposal
- Researched and implemented Hugging Face libraries and datasets
- Set up sentiment analysis
- Fine-tuned transformer's classifier
- Debugged and documented codebase