CSDS 395 - Report 1: An All-In-One NLP Stock Market Backtester

> Shaochen (Henry) ZHONG `sxz517` Jiaqi Yu `jxy618` Mocun Ye `mxy293`

Due and submitted 10/09/2020

Background

The background of the project remains largely unchanged from our perviously submitted proposal. So here we will provide a concentrated version of it.

Using NLP related approaches to do some kinds of prediction on stock market is nothing new among traders who want to develop profitable trading strategies, researchers who want to testify their models' performances, and also to every developer who wants to have some hand-on ML/DL experience.

Despite the popularity, we noticed that it is rather hard to verify a NLP-stock-prediction model's performance since the researcher will have to gather the plain text data, gather the company data, gather the stock market data, and categorize them in a way that is communicable with each other and the model; then the researcher will need to build a virtual trading platform that keeps track of all the trading signals generated by the model, and visualize them for evaluation.

To implement all these steps from ground up, it is required for a researcher to have certain level of proficiency on skills which are, from a research stand-point, fairly deviated from the nature of the NLP model itself (like scraping a website and understanding the fundamental mechanism of trading in stock market). Even though there are some very mature tools being developed in the subareas of this task (especially on the stock market backtesting area), it still requires a reasonably large amount of effort to couple them together, and to store necessary information in a way that are not only communicable with each other, but also suits to the design of each and every tool a research chose to use. It is our understanding this kind of preliminary work will distract a researcher from the essence of his/her work -- developing an SOTA model, and will also create a unnecessary barriers for researcher who wants to quickly testify an idea in a controlled manner, or to who are in the need of reproducing a published work.

We like to build a set of tools that may automate such process to a certain degree. The ideal workflow we visioned is that developers may import their plain text and company data in a certain format (or even use the build-in API to obtain such data, of course, with limitation on available channels), then we will have a set of functions (or parsers) available to execute and register the trading signals generated by a desired model; our toolkit will also able research to restructure data in a way that is suitable for his/her model. In our pervious proposal we said if time's available, we may even built in some classic NLP model just to provide a benchmark reference on "the same playing field," or develop my own HMM model I am researching right now for demo. **By re-evaluating of our team members' course load per TA's feedback, we will likely drop the idea of actually providing a technically significant model, but just to place a "dummy model" to demonstrate our project's capability.**

In our pervious proposal, we also researched couple related works regarding this task (mostly focusing on the fields of backtesting, obtaining stock market data, and obtaining plain text data). We analyzed the pros-and-cons of several existing public available libraries and described why are (or why aren't) we developing own our tools, and what functionalities were expected. Please do review our *Proposal* if you'd expect more information regarding related works.

Progress Report

Per our proposal, by the time this report is written, we should be working on:

- *Week of 09/28/2020: Refactor scraper / Technical selection on backtester / Learning for visualizer / Learning stock market basic*

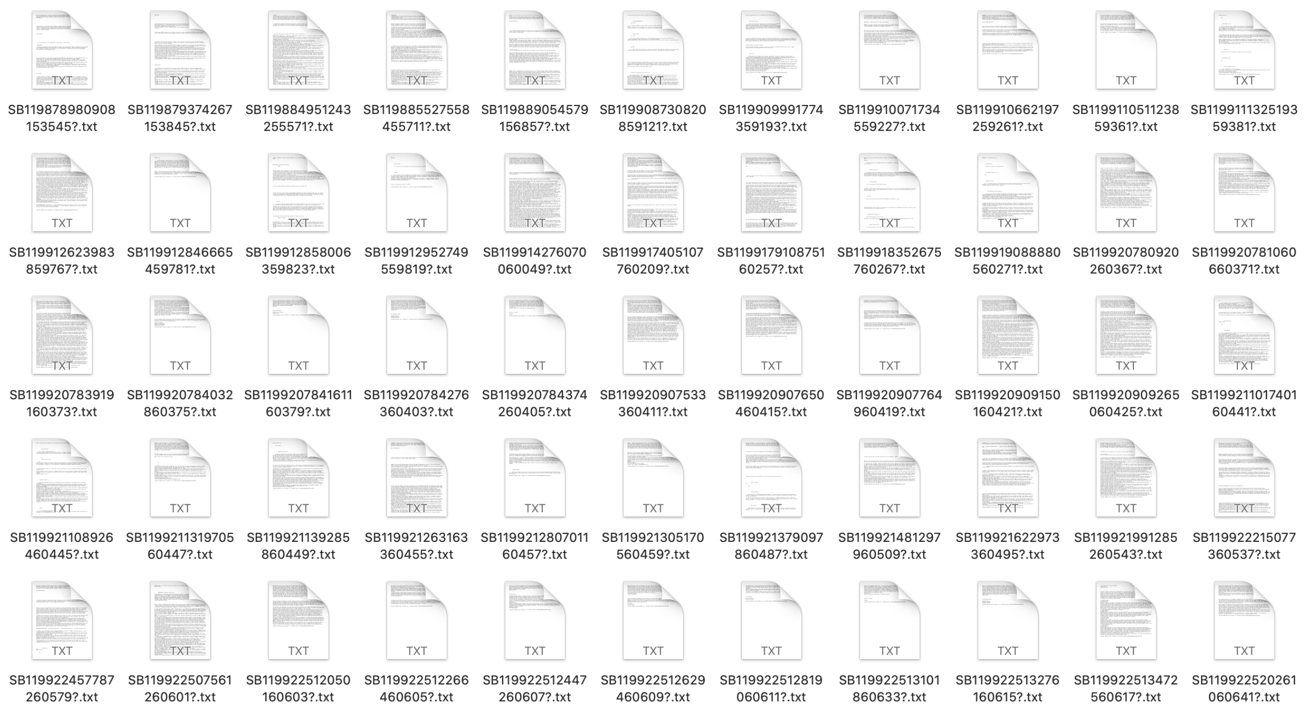
In an overview, we have followed our plan to a very exact manner. Although due to having to write this report, to prepare for upcoming midterms, and to handle some heavy deadlines on 10/5, we foresee a postponement our future plan for around an 1.5 after this week. We remain positive on delivery our project due to the redundancy we left, and we can also scale down the front-end part if necessary.

Regarding Text Input

As mentioned in the background session, one of the main focuses of our project is to increase the obtainability of high-quality, ease-to-use plain text data with reach metadata provided; so that a researcher may concentrate on the model without having to deal with the minute details of data obtaining/processing/basic cleaning.

As a proposed option and now and per TA's feedback, we decided to limited our provided "exemplary" plain text data API to be The Wall Street Journal as one of our group member (Henry) has perviously worked on it.

What we have in hand now is a set of scripts which capable of scraping almost all (there are couple of exceptions, i.e. this comic has no plain text information and our script will simply skip it) WSJ articles during a certain timeframe, and register mentioned companies market information accordingly. As a visual aid of the former achievement, the scripts we have are able to extract the below information from WSJ:



```
SB1198789809081535452.txt  SB1198793742671538452.txt  SB1198849512432555712.txt  SB1198855275584557112.txt  SB1198890545791568572.txt  SB1199087308208591212.txt  SB1199099917743591932.txt  SB1199100717345592272.txt  SB1199106621972592612.txt  SB119911051123859361?.txt  SB119911132519359381?.txt
```

SB119878980908153545?.txt | SB119879374267153845?.txt | SB119884951243255571?.txt | SB119885527558455711?.txt | SB119889054579156857?.txt | SB119908730820859121?.txt | SB119909991774359193?.txt | SB119910071734559227?.txt | SB119910662197259261?.txt | SB119911051123859361?.txt | SB119911132519359381?.txt

SB119912623983859767?.txt | SB119912846665459781?.txt | SB119912858006359823?.txt | SB119912952749559819?.txt | SB119914276070060049?.txt | SB119917405107760209?.txt | SB11991791087516025 7?.txt | SB119918352675760267?.txt | SB119919088880560271?.txt | SB119920780920260367?.txt | SB119920781060660371?.txt

SB119920783919160373?.txt | SB119920784032860375?.txt | SB11992078416116037 9?.txt | SB119920784276360403?.txt | SB119920784374260405?.txt | SB119920907533360411?.txt | SB119920907650460415?.txt | SB119920907764960419?.txt | SB119920909150160421?.txt | SB119920909265060425?.txt | SB11992110174016044 1?.txt

SB119921108926460445?.txt | SB11992113197056044 7?.txt | SB119921139285860449?.txt | SB119921263163360455?.txt | SB11992128070116045 7?.txt | SB119921305170560459?.txt | SB119921379097860487?.txt | SB119921481297960509?.txt | SB119921622973360495?.txt | SB119921991285260543?.txt | SB119922215077360537?.txt

SB119922457787260579?.txt | SB119922507561260601?.txt | SB119922512050160603?.txt | SB119922512266460605?.txt | SB119922512447260607?.txt | SB119922512629460609?.txt | SB119922512819060611?.txt | SB119922513101860633?.txt | SB119922513276160615?.txt | SB119922513472560617?.txt | SB119922520261060641?.txt

The naming of files looks a bit "random" because all WSJ articles' url are structured with a prefix of `https://www.wsj.com/articles/` and a suffix of `SB...`. Like for Delta Petroleum's Stake Sale Eases Need for Cash - WSJ the link is `https://www.wsj.com/articles/SB119909991774359193?`), where `SB119909991774359193?` is the suffix. So we named our scraped articles with such suffix as it is a unique ID for all WSJ articles. We understood this discovery is probably unique to WSJ, and we may therefore implement another layer of ID system to make it cross-distributor compatible.

Another key feature where our project thrives is the ability to gather rich and necessary metadata along with the plain text information from an article. e.g. One of the essential need of NLP-based trading is to be able to link a plain-text mention of a company to such company's market data information. For example, in this above mentioned article Delta Petroleum's Stake Sale Eases Need for Cash - WSJ, two significant companies were mentioned, which are:

```
Delta Petroleum
Tesoro
```

But which ticker name, exchange center, and legal full name does `Delta Petroleum` associate to? Sometimes, the mentioned company name is entirely different to its registered name on stock markets: e.g. *Coach* the apparel company's is registered as *Tapestry Inc.* with a ticker of `TPR` under *NYSE* exchange.

WSJ provided a hyperlink per their mentions of significant companies, but it is always out-of-date (like this time it links to https://www.wsj.com/market-data/quotes/DPTR, which is a page that is no longer available). We (mostly Henry) did some heavy engineering (in short, we inspect google results with some restriction syntax to get back to the new WSJ market data page of such company to obtain market information from it. In this case, for `Delta Petroleum`, it will be: https://www.wsj.com/market-data/quotes/DLTA/company-people) on this and able to obtain most companies market information accurately. The result will be something like:

```
[
    {
        "ing groep": {
            "market_data_url": "https://www.wsj.com/market-data/quotes/ing",
            "ticker": "ING",
            "exchange": "(U.S.: NYSE)",
            "legal_full_name": "ING Groep N.V. ADR",
            "cap_name": "ING Groep"
        },
        "baidu.com": {
            "market_data_url": "https://www.wsj.com/market-data/quotes/bidu",
            "ticker": "BIDU",
            "exchange": "(U.S.: Nasdaq)",
            "legal_full_name": "Baidu Inc. ADR",
            "cap_name": "Baidu.com"
        },
        "barclays": {
            "market_data_url": "https://www.wsj.com/market-data/quotes/bcs",
            "ticker": "BCS",
            "exchange": "(U.S.: NYSE)",
            "legal_full_name": "Barclays PLC ADR",
            "cap_name": "Barclays"
        },
...
```

(Note the above demo is up-to-change during the course of developing.)

One other key metadata is about the article itself: the title, the author, the exact publishing time, which session does it belong to (extremely critical when it comes to politics, e.g. a new report v. a column piece). We will able to register such metadata on the fly when scraping the plain-text information of the article.

Note a lot of these work were done perviously by one of our group member (Henry) under the help of another group member (Mocun). However, most of the scripts were written in a one-off fashion, thus very "hacky" and lack of usability. So we had, are, and will be putting heavy effort to refactor these scripts and hoping to have something like:

```
obj.get_plain_text_data (<start_time>, <end_time>, publisher = 'WSJ', credential dict =
publisher_token['WSJ'], restriction_dict = None, metadata_dict = None, output_path =
<default_dir>).
```

We except to finish the refactoring / adding new features to this segment for around one or two weeks. Modularity, usability, and a layer to support other publishers are our main focus.

Regarding Trading

As stated in the proposal, we are trying to design an integrated solution that has built-in tools for data manipulation while also works as a backtesting platform. Upon further study in the stock trading, we now have a more detailed design of the trading mechanism of our solution. The design of our system is built on two principles, the simplicity and the scalability.

In terms of the simplicity, we want our system to provide as simple and integrated interface as possible to the user. In our study of the existing backtesting platforms, we found most of them have complex and inefficient interfaces. Python-Backtesting, as an example, lacks a well-defined interface for the users to program their model. To obtain the moving averages or the relative strength index for a time period, the user would have to write their own averaging and computing methods, let the model keep the stock price info on each trading event, and do the computation at some point within the model. Also, the name of the method to obtain the trading data various and looks unpredictable. These are what we are trying to avoid in our trading model.

As for the scalability, we want our platform to also be able to accommodate models of a wide range of complexity, that is, models working on different time frame, adopting different range of trading event and information, and obtaining data either by batch or per time frame. Again, Python-Backtesting as the example, does not provide flexible time frame and the model would thus have to resample the data by itself. Also, most of the existing backtesting platforms only offer the most simple stock price data to the model, either discards or pre-processes the other stock events, such as splitting, merging, suspension, etc. We expect our platform to provide all the possible information in the original stock data to the model, while making most of them optionally adopted by the model. In this way, the user could focus only on what information their model need, and fetch them in the easiest way.

With these design objectives in mind, we so far propose a backtesting interface as follows. These designs may subject to changes as we actually implement them, but we do not expect the final product to be any more complex than the design we have now.

The usage of our backtesting platform is divided into three phases, the configuration phase, the data pre-fetching phase, and the iteration phase.

In the configuration phase, the user would initialize an instance of the backtesting platform, let it load the data, and do the configurations if necessary. The first two steps would be as simple as initialize a class with a path to the data folder. As for the third step, we define a series of functions and variables in the class to be configured:

Function/Variable	Parameters/Value	Description
`.time_resolution`	integer, in minutes	The time interval that the trading data is aggregated and provided to the model.
`.time_start`	integer, UNIX timestamp	The time that the trading emulation iteration starts.
`.time_end`	integer, UNIX timestamp	The time that the trading emulation iteration ends.
`.set_features`	*list_of_features: enumerators	Set the list of features that need to be provided to the model. This determines which features would be included when the platform calls the callback function of the model. See the third phase below for more about the feature.
`.set_init_callback()`	callable	Provides a function to be called by the platform when it starts to iterate through the data and emulate the transaction. This function is optional.
`.set_iterate_callback()`	callable, should accept a dictionary	Provides a function that the platform calls on each iteration so as to provide data to the model. See the third phase below for more on this function.

The second phase for the model to load the data, is actually optional. This is designed for the models that need to be trained, or would be more efficient if first trained, with batches of data first before it could be run online. We provides the following data fetching function:

Function	Parameters	Description
`.prefetch_data()`	start_time: integer, end_time: integer, *list_of_features: enumerators	Returns the data within the selected time interval with selected features. In real world applications, one don't have to call this function only once, fetch all the data into a table, and slice them by itself. Instead, it is suggested to call this function multiple times and feed the required data directly into the algorithms. For the list of features, see below.

Finally, in the last phase, the model should have been completely initialized and ready to be emulated on the backtesting platform. The model should then provide callback function to the platform so that the platform could run the model on each iteration of the emulation. We also provide a simple interface for the model to do the tradings, but it is suggested for the model to keep the trading information within itself.

Function	Parameters	Description
`.run()`	None	Starts the emulation. The emulation would be done from the `.time_start` to `.time_end`, with an interval of `.time_resolution`.
`.trade()`	stock: string, amount: integer	Do the trading on the selected stock with the desired amount. The amount could be both positive and negative for buying or selling.

To provide the data back to the model, we call the callback function with a dictionary, of which the keys are predefined in the platform, and can be selected with the functions mentioned above. In this way, a model could choose which information it needs, and would then get a simple and comprehensive list of those information on each iteration. Also, with the uniformly defined names of these features, the model developer would find it easier to work with these data features.

As for the list of the features, since we are still studying the stock market, we have not finalized it yet. Below is an incomprehensive list of the features that we would likely to provide.

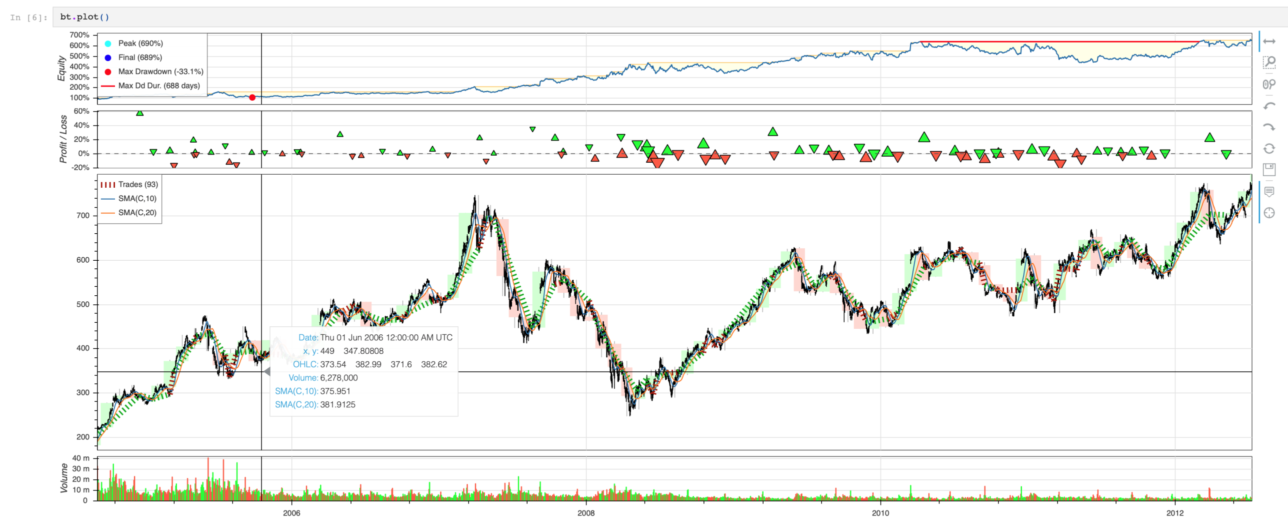
Name	Type	Description
`name`	string	The name of the stock. This feature is always provided.
`timestamp`	integer	The timestamp (or starting timestamp for a wider time frame) that this data happens.
`Quote`	float	The price (or, in most cases, the averaged price), within the time frame.
`Quote_last`	float	The price at the last minimal available time unit within the time frame. This is provided for more accurate trading behavior.
`Bidding_buy`	list	The descending sorted list of the latest buying bids within the time frame.
`Bidding_sell`	list	The ascending sorted list of the latest selling bids withing the time frame.
`Time_skip`	integer	When a time skip occurs, provides how many time units have been skipped in the data. This usually happens when date changes, or when suspension occurs.

Name	Type	Description
`Suspension`	boolean	Indicates that a suspension has occurred during the time frame. The flag would be kept True until it reaches a time frame within which the suspension is cancelled.
`Split`	float	Be none-zero when a split occurs in such stock, indicate how many shares one share is splited into.
`Merge`	float	Be none-zero when a merge occurs in such stock, indicate how many shares would be merged into one share.
`Ownership_Change`	list of dictionaries	Indicates the changes of the ownership of some stock by some other companies (owners). The information is provided in a list of dictionaries, which includes the names, current share, and changed share of that company (owner).

Regarding Visualization

We haven't done much actual development on visualizer as anticipated in our proposed plan. This is because the a visualizer is only developable after knowing what attributes are needed to be visualized and in what format will the visualizer module achieve such attributes (in our case, it will mainly be the `trade_log`).

We (mostly Mocun) did, however, inspected and duplicated some visualization results demoed by some popular backtesting library. e.g. the `backtesting.py`.



```
In [6]: bt.plot()
```

For a honest evaluation, with the ability of duplicating this kind of visualization, for the "overlapped" attributes between our need and the support of this visualizer, we will likely not "reinvent the wheel" due to the fact this design is good enough (if not any better). We will find a way to incorporate (or separately provide) our visualization on some NLP-only data, e.g. `word-freq`.

We have also experimented the idea of exporting a `TensorBoard`-like file which is interactable in browsers, as one of our team member Mocun has some related experience on this in his pervious project.

Also as encountered in a pervious RL project which Henry have contributed, we also looked into the possibility of visualizing model's "thought" in a sense of tracking some key attributes of the model (like this). We have tried some "playground code" on this with some dummy input, the coding is not too hard but it is hard to have different attributes tacked altogether while making them visually "synced" with a time series axis. We will reevaluate if we'd like to implement this feature after the trade log of Jiaqi is entirely designed and fixed. Meanwhile, Mocun will keep experimenting/duplicating & learning some demos or libraries so he will have the necessary skill to implement a visualizer once its dependencies are ready.

Future Plan

Our general goal remains largely unchanged from the first proposal. So we will still (at least) provide a plain text data provider, a stock market trader (obtaining stock market data + backtesting), and a visualizer. We will reasonably scale down or up on our implementation regarding the features on model-friendly data manipulation, complexity of visualization, and technical advancement of the demo model (for now, it is mostly likely going to be "dummy" model for the pure purpose of demonstrating the capability of the project).

Other then the above mentioned segments — which are mostly tasks related to coding — we understand the maturity and usability of a tool is heavily depended on its provided documentation and manual. Thus, we will allocate time to provide an user manual like this (which one of our group member have previously done for another project) to facility our potential users.

Please refer to the **Updated Management Plan** section for a more detailed timeline for future plan.

Updated Management Plan

- Week of 10/05/2020:
  - Report 1 writing.
  - Management plan planning.
  - (Reduced workload due to midterm preparation and Report 1 writing)
- Week of 10/12/2020:
  - Refactor scraper.
  - Virtual trading platform development / Regulate trade log format.
  - Design visualizer.
  - (Reduced workload due to midterm preparation)

- Week of 10/19/2020:
  - Finish plain text data provider development.
  - Visualizer development with respect to trade log.
  - Data organizer design.
- Week of 10/26/2020:
  - Test and debug coupling between virtual trading platform and plain text data provider.
  - Visualizer development with respect to trade log.
  - Data organizer development with respect to achievement of virtual trading platform.
- Week of 11/02/2020:
  - Visualizer development (if needed).
  - Test and debug coupling everything.
  - Report 2 writing.
  - (Reduced workload due to Report 2 writing)
- Week of 11/09/2020:
  - Dummy demo model development.
  - Test and debug coupling everything.
  - Paper writing.
- Week of 11/16/2020:
  - Documentation and manual writing.
  - Redundancy.
  - Paper writing.
  - Post making (if needed).
  - (Reduced workload due to Paper writing)
- Week of 11/23/2020:
  - Redundancy.
  - Final report writing.
  - Polishing our delivery.

Contribution Acknowledgment

- **Henry ZHONG**
  - Developed WSJ scraper script with the help of Mocun. `code`
  - Refactoring WSJ scraper script with Mocun. `code`

  - Designing wrapper layer on the plain text data provider module with Jiaqi. `design`
  - Drafted **Regarding Text Input** section of *Progress Report 1*. `report`
  - Wrote **Background** section of *Progress Report 1* with contribution of Jiaqi. `report`
  - Wrote **Future Plan** and **Updated Management Plan** sections of *Progress Report 1*, however the it is just a reflection of our team discussion. `report` `management`

- **Jiaqi YU**

  - Designing wrapper layer on the plain text data provider module with Henry. `design`
  - Researched the implementations of necessary components of stock market trading. `design`
  - Educating other group members the basic of stock market trading to help design the interface between different modules. `design`
  - Drafted **Regarding Trading** section of *Progress Report 1*. `report`
  - Wrote **Background** section of *Progress Report 1* with contribution of Henry. `report`
  - Contributed to **Future Plan** and **Updated Management Plan** sections of *Progress Report 1*. `report` `management`

- **Mocun YE**

  - Experimenting basic of of data visualizer. `code`
  - Researched the design of data visualizer with respect to back testing platform. `design`
  - Researched the design of visualizer with respect to model feature representation. `design`
  - Drafted **Regarding Visualization** section of *Progress Report 1*. `report`
  - Contributed to **Future Plan** and **Updated Management Plan** sections of *Progress Report 1*. `report` `management`