Data Extraction and Sentimental Analysis from Equity Research Reports at Scale

https://github.com/PatternDetection/

Chenfan Zhuang, Xin Zeng, Qinyi Chen, Yujie Cai

May 6, 2021

Executive Summary

Written reports or articles by equity researchers in the finance industry can serve as important guidelines in building up secondary market investment portfolios. In the light of efficiency market hypothesis, information and stock price are related. The information summarized for a corporation relieved us from reading off lengthy financial statements, and suggestions to buy, hold, or sell from well-known institutions are reliable guides for institutional or individual investors. Ideally, equity researchers use their information and analysis to lubricate the buy- and sell-side of the market. However, the information conveyed by these reports is not utilized efficiently enough. First of all, the large amount of reports produced everyday has the potential to overwhelm investors. Investors may have trouble choosing the appropriate set of reports because human's reading ability is capped. Moreover, an equity research report can last for over 10 pages, full of text and graphs. Some information thus will be overlooked, or ignored, leading investors to the non-optimized decisions.

Still, investors have strong incentives to incorporate the information covered in reports into their investing strategies, given lots of quantitative strategies call for market sentiment data. These phenomena are prominent in the China A-share market in particular. As an emerging market, the A-share market is on the way to regularization and efficiency. The equity research business and quantitative finance are still in their developing stage. Therefore, the market calls for a more efficient usage of the information.

Therefore, we aim to use modern computer science techniques to identify, extract, and utilize the financial economic information/data on equity research reports written for A-shares. That is, we turn equity research reports into structured data for financial analysis. The project demonstrates the use of computer science and data science techniques, namely (1) computer visions, and (2) natural language processing(NLP). In short, our project pipeline starts from layout detection on equity research reports, then goes to OCR to extract the text data from equity research reports. Then we use NLP to further extract the important market sentimental feature for analysis afterward. Finally, we combine the sentimental features and other numerical features as predictive signals into our backtesting models on stock performance forecasts.

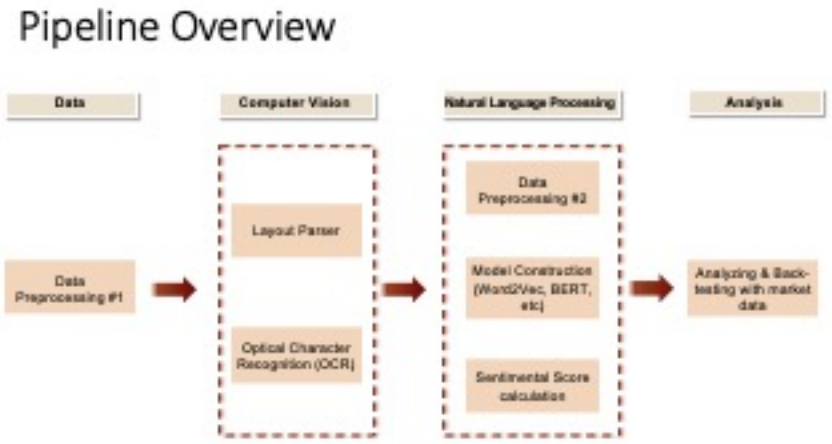


Figure 1: Project pipeline overview

Previous research and existing models in CV and NLP have provided us with some directions to start with. The Harvard Layout Parser is a good starting point for us because of its good performance and friendly API [8]. As for the NLP task, we start by working with Word2Vec and several neural network models, and this approach is quite common in the academia and the industry[4, 6]. The detailed reasoning of model selection and experiments was provided in the main text of the report.

We not only creatively combine the publicly available tool to accomplish our task of turning equity research reports into structured text data, but also make improvements on the CV and NLP model we have built upon. In the end, in CV part, we define our own performance matrix to evaluate and fine-tune the public model to achieve better performance of OCR; in NLP part, we fine-tune the FinBERT model and the Chinese BERT Model with our self-selected labels. In the end, we conduct the backtesting and discover the predictability of our sentimental factors constructed from the NLP.

There are few enhancements and future developments that are worth mentioning for our project. We could experiment with more resources in OCR. For NLP, we will try to construct a high-quality dataset with more data points, which would greatly improve the accuracy and efficiency of fine-tuning our Chinese BERT model.

The final report is organized in the following manner. Section I introduces the equity research report data, what is the structure and why it is hard but important to be transferred into structured data. Section II presents the CV procedure and results. Section III presents the NLP procedure and results. Section IV shows the effectiveness of financial analysis using the data we extracted from equity research reports. The last section V focuses on the future and possible improvements.

1 Data

1.1 Data Description and Retrieval

What is our data?

In finance, equity research reports are typed articles by equity researchers featuring the analysis of certain publicly traded companies. Most of the time, they serve as important guidelines in building up secondary market investment portfolios. In the light of efficiency market hypothesis, information and stock price are related. The information summarized for a corporation relieved us from reading off lengthy financial statements, and suggestions to buy, hold, or sell from well-known institutions are reliable guides for institutional or individual investors.

The above applies to almost all financial markets around the world. However, in different markets, the reports might have different format conventions. In our project, we are dealing with equity reports written in Chinese. When retrieving the reports, they are usually in pdf format. Here is our example:

The first page. The green box on the top shows the research provider(i.e. who write the report), and usually there is a publish date. The blue boxes are titles. The upper one shows the name of the equity being researched, and the lower one shows the title of the report. For instance, in this report, the name of the company is Fujian Torch Electron Technology Co., Ltd. The title says "Informatization recommendation: both production and sales are booming, and our first coverage rates the equity 'outperform the industry' ". Below these, the report usually summarizes the researcher's core opinion about the firm in the red box. Those core opinions will be explained in great detail starting from later pages. Other supplementary information such as key performance forecasts and price trends are shown in the yellow boxes.

Figure 2: First page of equity research report

The middle pages. After going through the content, we enter the main text of the equity research report. The example continues as the following: we are able to see that this section is a combination of dense texts, graphs and tables. The middle pages present the researcher's main reasoning on why he or she is recommending company or not, with detailed explanation.

The end. The ending pages are disclaimers and contact information.

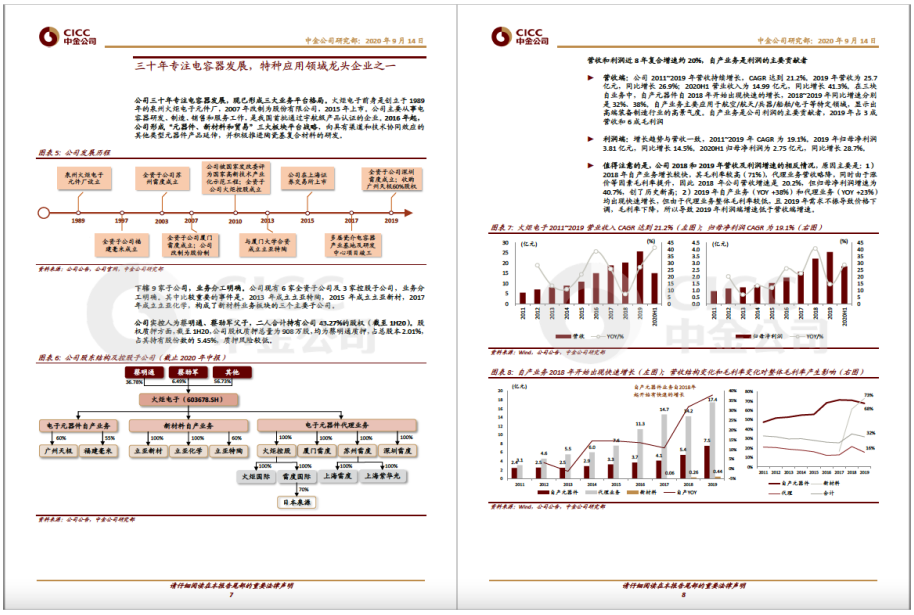


Figure 3: body page of equity research report

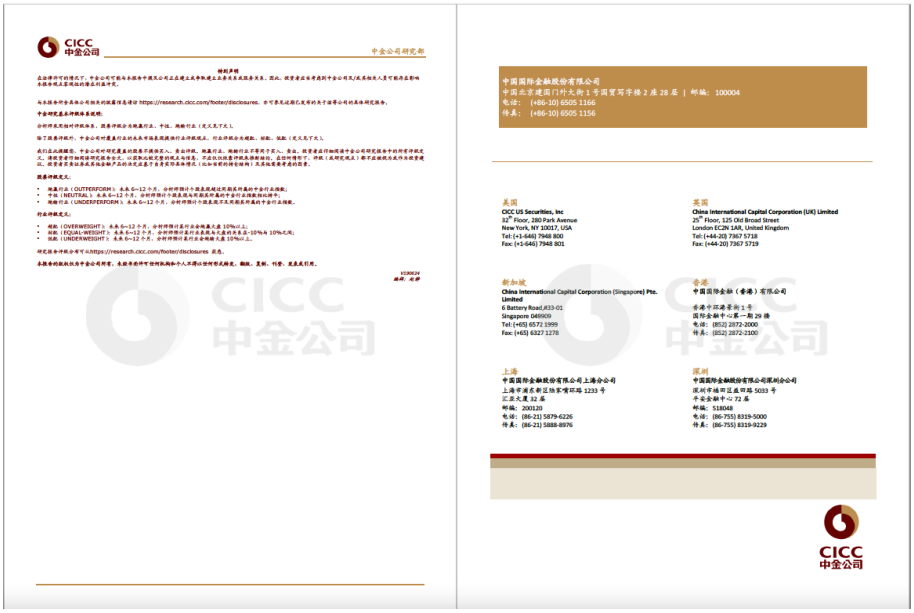


Figure 4: ending page of equity research report

The length of the equity research report can vary from 3-4 pages to over 30 pages. Without the disclaimer and contact information, the useful text ranges from 1-2 pages to around 25 pages. How-

ever, there is still too much information to capture if we want to process the whole report. Instead, we put ourselves in the position of investors who need to read many reports to make investment decisions. Usually, investors only want to read off the gist of the length report. Hence, they tend to focus more on the first page, since later pages are providing detailed information to support the opinion made in the first page. Also, as we can see, the equity research reports arrange text, table, and graph together. For computers, it is a tremendous task to separate one by one if we are about to analyze the whole report. Therefore, to get the maximum efficiency for both human and computer, we will only extract the information from title and text on the first page. We argue that this information is representative of the full-length report and will be useful for the investors.

How to retrieve?

Since we will process equity research reports in pdf format as our raw data, we need to retrieve a good amount of them. The equity research reports on Wind and Bloomberg are generally available to the general public for free. However, certain companies will restrict the non-institutional investors to view the report. For this project, we exclude any restricted report, which is defined as the report which cannot be found on `https://robo.datayes.com/`(a zero-cost, semi-professional, public financial data platform for everyone). Using semi- and professional platforms such as Wind and Bloomberg and Datayes, we successfully retrieve the public equity research reports and stock trading data, like price and volume. The equity research reports are our major subject to process, and the stock trading data will be used in the later step in backtesting.

We obtained 628 public equity research reports on 153 stocks in Shanghai and Shenzhen Exchange. Also, we have downloaded corresponding stock trading data, like price and volume, from 1/1/2011 to 2/11/2021. Since manually downloading reports have required a large amount of efforts, given this is not the key focus in our project, for this stage, we think these reports suffice our package development.

Because we save the report in a manner of "XXXXXX_YYYYMMDD" (stock code + date of the report), we can retrieve the time and code data from file names. To sum up, we will have code, time, opinion/sentimental data from the data processing in layout detection and OCR.

1.2 Data Pre-processing

We implement a pre-processing for the pdf data. We use the "pdf2image" (`https://pypi.org/project/pdf2image/`)library to convert PDFs to JPEGs with dpi=200, which experimentally works well as the input of the layout parser. The method to batch convert PDFs is located in our github repository.

1.3 Data Labeling

We discover that useful information is mostly embedded in the title and the text section, on the first page of equity research report. Therefore, we label our image with two sections: title and text. For 628 reports, we randomly selected 199 reports to be our test set, and the rest 429 reports are used for training in our later fine-tuning.

2 Layout Detection

This part aims at detecting layout of equity research reports which will be used as the input of OCR to improve the OCR quality. We know there are some OCR tools that already involve the layout parsing explicitly or implicitly. For example, the commercial OCR - Baidu API we use (details can be see in section 3), provides the location. But we prefer to make layout detection as a single step, because:

1. simpler input for OCR can yield better results. Feeding the whole image could output very noisy OCR results, hurting the effect of the NLP module.
2. we want to study the performance drop caused by domain gap (dataset difference) in layout detection and see for a new type of data if we can apply the public models without fine-tuning to get satisfactory results?

As mentioned above, we focus on detecting the regions of the title and the text, and then crop them out as the input for the OCR module. There are many open source layout parsing methods on the Internet. We choose Harvard Layout Parser [8] because of its outstanding performance and friendly API. We applied the public models to our dataset with post-processing and compare the performance with a fine-tuning model on metrics devised for this scenario.

2.1 Self-Defined Performance Metrics

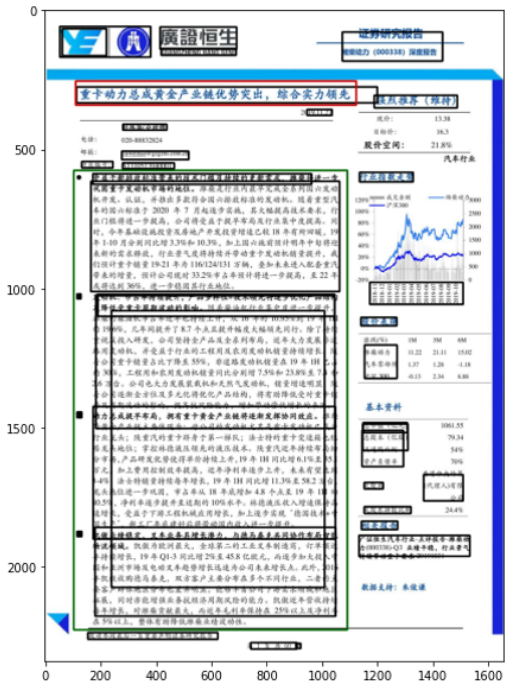
The traditional metrics used in Object Detection like mAP [2] is not suitable here since the boxes detected out are used for OCR extraction. Therefore, tightness of the prediction can be considered less as the coverage is more important. In other words, big is better than small, but not too big. Besides, for the text region, the open models tend to predict a series of bounding boxes to cover each paragraph, while the ground-truth that human labeled in our dataset is commonly a single one covers the whole text. For OCR, both fine and coarse boxes are good to use. To fairly evaluate the practicability of the public models, we propose a new performance metric.

Since we allow multiple boxes together used to cover one target, we count "fusion IoU" instead to see the coverage of the fused predictions 5b.

The algorithm to calculate "fusion IoU":

1. For a given rect R (groundtruth), find the bounding boxes (prediction) C whose intersection with R is greater than a threshold T, for instance, for text, T=1% area of R; for title, T= 20% area of R.
2. Calculate the union of C, the result is called fusion F;
3. Calculate the IoU between R and F.

Since we believe that the title and the text usually contain the keywords for later sentiment analysis, we hope they can be covered in most cases. So we define the target is "covered" if its fusion IoU > 0.8. We wonder the proportion of the "covered" cases, i.e. "cover rate" of the dataset.



(a) Red: title (groundtruth), Green: text (groundtruth); Black: raw prediction from "PrimaLayout-mask_rcnn_R_50_FPN_3x" model

(b) Blue: merged prediction for title; Pink: merged prediction for text

(c) Red: title (groundtruth), Green: text (groundtruth); Black: filtered prediction

(d) Red: title (groundtruth), Green: text (groundtruth); Black: filtered prediction

Figure 5: Layout Detection Result

If the "cover rate" $> 80\%$, we can say the model is on a level that ensures the characters are almost included in without involving too much noise.

The stats for the public models and the fine-tuned one will be listed in section 2.3.

2.1.1 Public Models

The source of images and the type of labels cause a big performance drop in our dataset. According to our observation, the model mask_rcnn_R_50_FPN_3x trained on Prima [1] Dataset is the best among all the publicized. So we will focus on this model to design the post-processing method.

As Figure 5a shows, the black boxes are the model predictions with a low score threshold for retrieving more and with a category limit to discard the non-text regions, like figures or tables. According to our observation, the width of the text generally takes up at least half of the page. So we remove the boxes with width in percentage less than 0.5. The filtered results can be seen in Figure 5c and Figure 5d. Then apply the fusion method introduced in section 2.1 to count the "fusion IoU".

2.2 Fine-tuning

Fine-tuning is a process of training a model from an existing checkpoint, or a model weight initialization arrived at through previous training.
We use Detectron2 [9] to fine-tune the mask_rcnn_R_50_FPN_3x model pretrained on the PrimaLayout dataset (reffered to as "PrimaLayout-mask_rcnn_R_50_FPN_3x" hereafter) because the "core" of Harvard Layout Parser is Detectron2. Detectron2 is Facebook AI Research's software system that implements state-of-the-art object detection algorithms. Detectron2 provides a series of training and testing procedures for a series of deep learning algorithm, including object detection, instance segmentation, and human pose estimation.
The advantages of detectron2 are mainly reflected in the following three points:

- The R-CNN series is the strongest implementation. After all, the R-CNN series is in FAIR, and it also supports new features, such as panoptic segmentation and rotated bounding boxes.
- It is highly modularized and extensible. Specifically, detectron2 can be regarded as a basic library. When implementing a new model, we use import rather than modify.
- The training speed is faster.

Based on the training config of PrimaLayout-mask_rcnn_R_50_FPN_3x, we adjust some key parameters to fit our data scale and GPU resource - we train it on Google Colab notebook (https://colab.research.google.com/notebooks/intro.ipynb):

```
cfg = get_cfg() # get default config file
cfg.merge_from_file("/content/drive/MyDrive/colab_files/Documents/config.yaml")
# Download from https://www.dropbox.com/s/yc92x97k50abynt/config.yaml?dl=1

cfg.DATALOADER.NUM_WORKERS = 4
cfg.MODEL.MASK_ON = False
cfg.MODEL.WEIGHTS = "/content/drive/MyDrive/colab_files/Documents/model_final.pth"
```

```
# https://www.dropbox.com/s/h7th27jfv19rxiy/model_final.pth?dl=1

cfg.SOLVER.IMS_PER_BATCH = 8 # batch size, tune in conjunction with SOLVER.BASE_LR
cfg.SOLVER.BASE_LR = 0.0002 # pick a good LR
cfg.SOLVER.MAX_ITER = 8000
cfg.SOLVER.WARMUP_ITER = 1000
cfg.SOLVER.STEPS = [5000, ]     # decay learning rate at 5000
cfg.SOLVER.GAMMA = 0.1
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 2 # we have two labels: title and text
```

- To specify config file, we firstly get default config file, and then use "cfg.merge_from_file" to merge in custom params. Here we introduce "PrimaLayout-mask_rcnn_R_50_FPN_3x" config file.
- "cfg.DATALOADER.NUM_WORKERS = 4" means we have 4 data loading threads.
- We turn the "cfg.MODEL.MASK_OFF" because in our case, the bounding box can be treated equally as the mask. And under the limited GPU memory size, the existing of the mask branch would further decrease the batch size and slower the training speed. Therefore we choose to turn it off. The result of fine-tuning the mask_rcnn model without mask branch absolutely meets our expectation.
- "cfg.MODEL.WEIGHTS" is the model initial weight.
  Here we use "PrimaLayout-mask_rcnn_R_50_FPN_3x" as the pre-trained model.
- "cfg.SOLVER.IMS_PER_BATCH = 8" is the largest batch size we can set at google colab. If we increase the "cfg.SOLVER.IMS_PER_BATCH" higher than 8, there would be a "RuntimeError: CUDA out of memory" as the picture 6 shows.

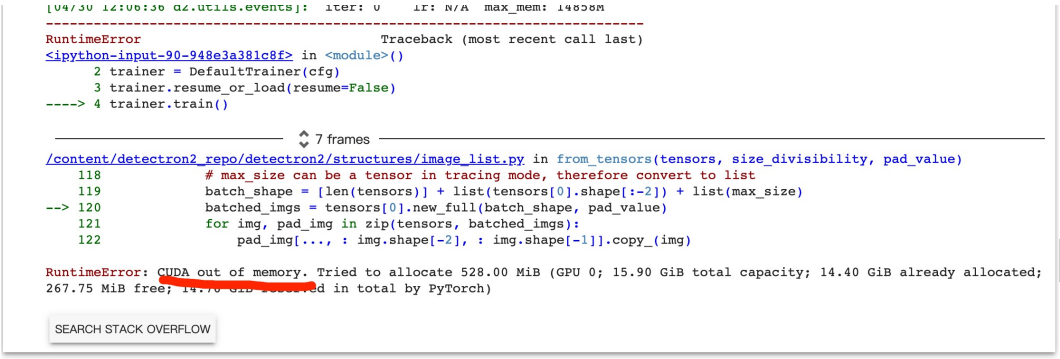


Figure 6: Problem: CUDA out of memory

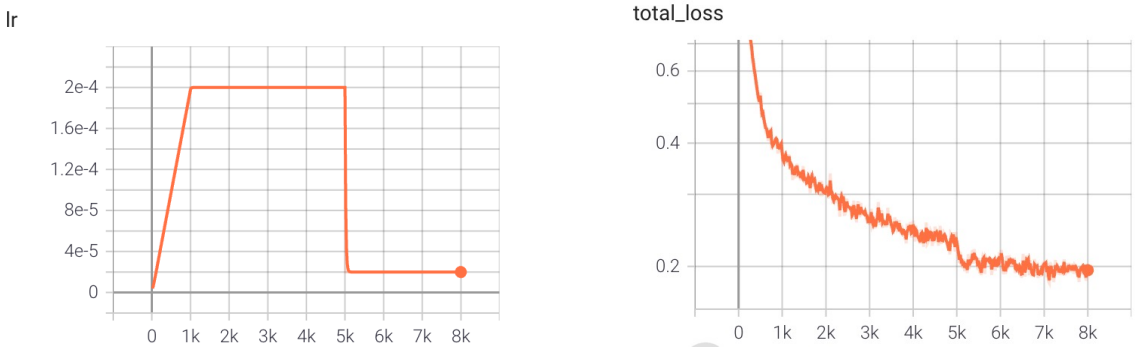
- "SOLVER.MAX_ITER" means Total iterations. Detectron2 don't use epochs, but we can calculate the number of epochs using the following equation:

$$number\ of\ epoch = \frac{SOLVER.MAX\_ITER \times SOLVER.IMS\_PER\_BATCH}{len(images)}$$

In order to get a good model, we need at least 10 epochs. Therefore,

$$SOLVER.MAX\_ITER \geq \frac{number\ of\ epoch \times len(images)}{SOLVER.IMS\_PER\_BATCH} = \frac{10 \times 429}{8} \approx 536$$

- Including a "warmup" in our LR schedule can help overcome some routinely empirically encountered optimization difficulties. "SOLVER.WARMUP_ITERS" is the number of training iterations, during which the LR will linearly grow from zero with a slope defined by SOLVER.WARMUP_FACTOR (default = 0.001). In our case, SOLVER.WARMUP_FACTOR is set to the reciprocal of SOLVER.WARMUP_ITERS, so the LR linearly grow from zero to SOLVER.BASE_LR in the warmup phase.
- Optimization in Detectron2 is controlled by "solver" hyperparameters. In order to prevent the learning rate from being too large, it will oscillate back and forth when it converges to the global best point. Therefore, We use "cfg.SOLVER.STEPS ", a list of iterations at which Learning Rate will be "stepped down" by factor of "cfg.SOLVER.GAMMA".
- "cfg.SOLVER.BASE_LR" is the base learning rate. We have observed in the experiment that the loss value will continue to decrease when the learning rate is around 0.0002. Picture 7a show the changing process of learning rate in our model training.
- "cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE" indicates the number of ROI's (Region of Interest) per training minibatch. Se the "cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE= 128" is faster, and good enough for our dataset (default: 512)
- There are two labels in our dataset: "title" and "text".
  So number of classes ("cfg.MODEL.ROI_HEADS.NUM_CLASSES") = 2.
- Train loss is the loss on the training data, which measures the fitting ability of the model on the training set. When a model has finished running, the output folder will contain a metrics.json file which can be used to plot loss metrics. From picture 7b we can observe the convergence of train loss (total_loss from 1.391 to 0.1976).



(a) Learning rate during the training

(b) Total_Loss during the training

Figure 7: Learning rate and Train Loss

(a) Orange: title (prediction), IoU = 85% ;
Red: text (prediction), IoU = 98%

(b) Yellow: title (prediction), IoU = 98% ;
Pink: text (prediction), IoU = 99%

(c) Green: title (prediction), IoU = 94% ;
Pink: text (prediction), IoU = 94%

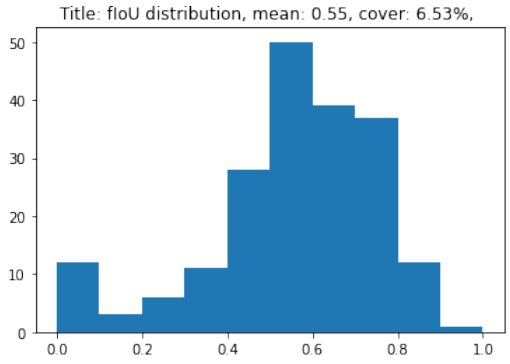
(d) Blue: title (prediction), IoU = 97% ;
Red: text (prediction), IoU = 99%

Figure 8: Fine-tuned Layout Detection Result

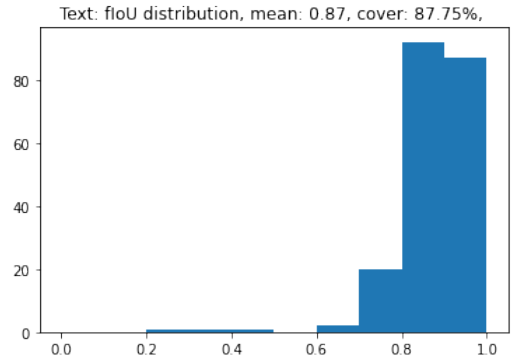
2.3 Comparison and Conclusion

Table 1: The mean fusion IoU(fIoU) and cover rate(cover%) on title and text of each model.

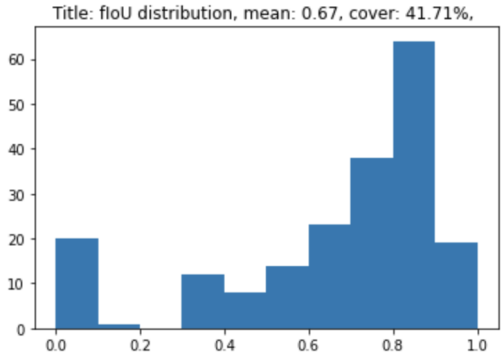
	title		text	
Model	mean fIoU	cover%	mean fIoU	cover%
HJDataset-faster_rcnn_R_50_FPN_3x	0.01	0.00	0.31	13.73
HJDataset-mask_rcnn_R_50_FPN_3x	0.00	0.00	0.21	7.84
HJDataset-retinanet_R_50_FPN_3x	0.00	0.00	0.19	8.33
NewspaperNavigator-faster_rcnn_R_50_FPN_3x	0.00	0.50	0.00	0.00
PrimaLayout-mask_rcnn_R_50_FPN_3x	**0.55**	**6.53**	**0.87**	**87.75**
PubLayNet-faster_rcnn_R_50_FPN_3x	0.45	4.52	**0.87**	86.27
PubLayNet-mask_rcnn_R_50_FPN_3x	0.37	5.53	0.84	81.37
PubLayNet-mask_rcnn_X_101_32x8d_FPN_3x	0.38	**6.53**	0.82	80.39
Fine-tuned PrimaLayout-mask_rcnn_R_50_FPN_3x	**0.67**	**41.71**	**0.94**	**99.02**



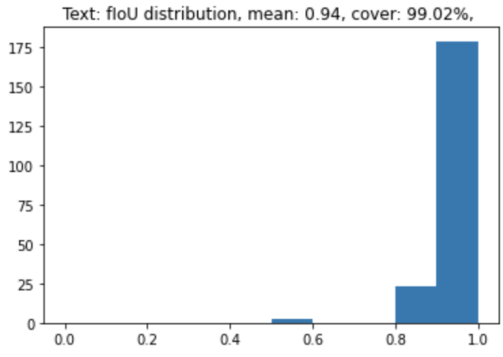
(a) Distribution of fusion IoU of title from PrimaLayout-mask_rcnn_R_50_FPN_3x



(b) Distribution of fusion IoU of text from PrimaLayout-mask_rcnn_R_50_FPN_3x



(a) Distribution of "fusion IoU"(fIoU) of title from Fine-tuned PrimaLayout-mask_rcnn_R_50_FPN_3x



(b) Distribution of "fusion IoU"(fIoU) of text from Fine-tuned PrimaLayout-mask_rcnn_R_50_FPN_3x

The best performance of the public models comes from model mask_rcnn_R_50_FPN_3x trained on the Prima dataset [1], outperforming slightly the series trained on the PubLayNet [10] dataset. The data and labeling in these two datasets are pretty "close" to our dataset, leading to a smaller "performance drop" compared with the other models trained on the HJDataset [7] and the NewspaperNavigator [5] dataset.

From the stats listed above, we can clearly see the great improvement of fine-tuning, from 6.53%(title cover rate)/87.75%(text cover rate) to 41.71%/99.02%, with 429 images only. For the practical usage, the combo of "public models + post-processing" is an economical option since the cover rate on text looks not bad and the text region usually contains the information that the title expresses. If someone can't access any GPU resource or valid dataset, using public models with appropriate processing could bring cost-effective results.

3 OCR

This module is used to convert the text blocks into words in txt format (non-text blocks will be ignored). The input of OCR are images and bounding boxes. And the output are corresponding text content of images with right order. We choose Baidu OCR API (`https://ai.baidu.com/tech/ocr/general`) to extract the words from the detected text regions. 11 shows a public demo of Baidu OCR. It's free for 50000 calls per day with QPS (max query per second) = 2.

Figure 11: Public Demo of Baidu OCR

Embed the code in our pipeline (`https://github.com/PatternDetection/Project/blob/main/notebooks/convert_pdf_and_parse_layout.ipynb`) and we get our words in txt format.

Figure 12: Sample Output from OCR

Baidu also provides the OCR API with location. Unfortunately this service limits 500 calls per day for free usage. We are going to sacrifice a level of precision to achieve the speed and the convenience without extra cost.

4 Natural Language Processing

In this part, we aim at extracting effective and useful sentimental information from the text content we get from OCR. We mainly contribute to the construction of the Word2Vec and a Fine-tuned Chinese BERT model focusing on the sentimental analysis of Chinese equity reports, providing an useful tool for researchers and investors to convert the text data in Chinese equity reports into sentimental signals in investment at scale. Moreover, due to the lack of high-quality sentimental datasets in both sentences and words with labels focusing on the analysis of Chinese equity reports, we combine a set of methods, like web crawler and manual labelling, to create such datasets. This not only benefits the performance of our Word2Vec and BERT models, but also provides useful information for further researchers.

4.1 Data Description

After OCR, we get 628 files with format of txt. Each of them contains the text information of the corresponding equity report. The files are named in a manner of "XXXXXX_YYYYMMDD_N"

(stock code + date of the report + unique identifier). The unique identifier "N" is used to distinguish the possible multiple reports for the same stock in the same date. One of the sample data can be seen in https://github.com/PatternDetection/Project/blob/main/results/OCR/600016-20200409-0.txt.

4.2 Word2Vec Model

4.2.1 Data Preprocessing

1. Since we desire to train our Word2Vec model to achieve more word vectors. For the generality, we choose zh-wiki which means the wikipedia in Chinese as our training corpus. We firstly download the data from https://dumps.wikimedia.org/zhwiki/latest/zhwiki-latest-pagesarticles.xml.bz, which contains all the Chinese wikipedia.

2. Then we implement a parse_zhwiki_corpus.py file to convert the format from xml to txt. Using the WikiCorpus method from the gensim package in Python, we could complete the conversion without decompassing the xml file.

3. Since Chinese is divided into traditional and simplified characters, and there only exist simplified characters in our equity research report. To make our input more uniform, we implement chinese_t2s.py file to convert traditional Chinese characters to simplified Chinese characters.

4. Since the file we get might contain some English words, we implement remove_en_blank.py to remove the English words and the space.

5. Lastly, we implement corpus_zhwiki_seg.py to do text segmentation. Unlike English, Chinese does not use blank to tokenize words. Therefore, before doing NLP over Chinese, we need to have one more tokenization step, which turns an article into a bag of words. For any given piece of text content, we could use the Jieba package (https://pypi.org/project/jieba/) for text segmentation

4.2.2 Word2Vec Model Construction

1. After we get the processed data, we use the processed data to train the Word2Vec model. To minimize the influence of parameters to our final outcome, we would train a series of models with different inputs of parameters (like Size=100, Min_count=5; Size=500, Min_count=4 and so on). We would test different combinations of parameters when generating our NLP vectors and doing sentiment analysis in the following procedures.

4.2.3 Calculation of Sentimental Score

1. We first collect some Chinese common words from various resources, like https://github.com/CodeBrauer/1000-most-common-words, https://github.com/oprogramador/most-common-words-by-language and so on. Then in the following parts, when we compute the Senti-score of an article, we will just need to look up the common word set. We select 862 common words in total for our sentimental analysis.

2. Then we build a sentiment lexicon to tell our model which words are positive or negative. We define a sentiment lexicon as a small tally set to evaluate the sentiment of a single word. We

aim to choose 50 words that have a positive attitude toward the market (positive words) and the other 50 have the opposite sentiment (negative words). For instance, the positive words include bullish, climb, surge, hortation and so on. The negative words include bearish, fall, slump, sanction and so on. We store them as a DataFrame in a csv file. Due to the lack of such high-quality sentimental labels focusing the Chinese equity reports, we take reference of multiple resources, like `https://github.com/MengLingchao/Chinese_financial_sentiment_dictionary`, and we select and refine the sentiment lexicon manually. Our final sentimental label can be seen in `https://github.com/PatternDetection/Project/blob/main/data/label/label_word.csv`.

3. Before computing the senti-score of equity research paper, we first compute both the morphological similarity and the semantics similarity between the label words and every single common word.

   Morphological Similarity: We calculate morphological similarity by training the Word2vec model. After training, we get to know the word vectors of all words. We just look up a word vector of a word in the model like a dictionary and calculate the morphological similarity of two words with their cosine distance.

   Semantics Similarity: We compute the semantic similarity of words using WordNet. WordNet uses trees to record words. The structure of trees defines distances naturally. We could use the shortest path linking the two nodes representing the words to compute. Take the reciprocal of the shortest path of the two nodes as the similarity. The similarity between a word and itself is 1. We also define the similarity between two words is 0 if there is no path linking them. With definitions above, the semantics similarity we compute will be a value between 0 and 1. The larger the value is, the more similar the two words are semantically.

4. With the computation above, we finally have a vector of 200 dimensions for every word. The first 100 dimensions are the Word2vec similarities with the 100 words in the sentiment lexicon. They show the morphological similarity with the label words in sentiment lexicon. The last 100 dimensions, on the other hand, are the WordNet similarities with the 100 words in the sentiment lexicon, which represent the semantic similarity with the label words. Since the words in the sentiment lexicon show attitudes toward the market, based on the similarities above, we can evaluate the attitude of a specific word, namely the sentiment of it. Figure 13 shows the results of morphological similarity and semantics similarity after this step.

	补助w2v	部署w2v	超过w2v	盈利w2v	成功w2v	成熟w2v	充分w2v	出名w2v	刺激w2v	大幅w2v	...	弱势wn	不利wn	严峻wn	流出wn	下降wn	抑制wn	罚金wn	困难wn	下挫wn
0	0.027832	0.035977	0.135615	0.103383	0.042817	0.098742	0.040337	0.014064	0.095846	0.071640	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.010589	0.011660	0.131764	0.069275	0.142454	0.062292	0.142887	0.125542	0.182542	-0.052107	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	-0.070025	0.018138	0.114243	-0.082113	0.076196	-0.022489	0.096041	0.183013	0.026723	-0.027708	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.191511	0.114508	0.232280	0.156647	0.192779	0.219695	0.280992	-0.026666	0.321580	0.322149	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.087248	0.054850	0.116088	0.145896	0.219925	0.116418	0.280501	-0.083467	0.215051	0.165106	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
857	-0.116461	-0.226354	-0.064629	-0.067584	-0.049404	0.035456	-0.053601	0.108326	0.007232	-0.123439	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
858	0.188595	0.220464	0.352763	0.320877	0.284385	0.180636	0.235561	-0.126035	0.033288	0.377693	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
859	0.069706	-0.067908	0.069624	-0.036048	0.118291	0.203487	0.061724	0.282720	0.079975	0.008252	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
860	-0.009357	-0.190153	-0.016503	0.008825	0.141174	0.172706	0.124342	0.242021	0.174417	-0.192497	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
861	0.116017	-0.030222	0.018876	0.006065	0.090704	-0.014079	0.052858	0.140999	0.070653	-0.002118	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

862 rows × 200 columns

Figure 13: Results of Morphological Similarity and Semantics Similarity

5. Then, we combine both morphological similarity and the semantics similarity to calculate the sentimental score for each common words. We firstly define the scores of positive words and negative words in sentiment lexicon to be +1 and -1 respectively. Then we use the 200 morphological and semantics similarities to calculate the weighted average, and we regard the weighted average as the sentimental score of a target common word. One additional note is that the words included in the COW is far less than words in the Word2Vec model we trained, we would get 0 in the semantics similarity between some common words and some labels. And we would ignore these 0 values during the calculation of weighted average. After this step, we have the sentimental score for every common words.

6. Finally, we compute the sentimental score of each text information from our equity reports by adding up the sentimental scores for words in the corresponding text.

4.3 Neural Network Model Comparison

Since Word2Vec model learns embeddings at "word" level. Such representations cannot generate vectors for words encountered outside the vocabulary space. Moreover, there are no shared representations at sub-word levels. Considering the drawbacks of Word2Vec model, we then compare several Neural Network models, like Fully Convolutional Network, Fully Convolutional Network with Embedding, BERT and Convolutional Neural Network with Embedding, to see their performance on sentimental classification. We use a high-quality dataset from Financial PhraseBank (`https://www.researchgate.net/publication/251231364_FinancialPhraseBank-v10`).

We can see that the BERT model achieves higher accuracy and lower loss with smaller epochs in our financial sentimental dataset. This motivates us to focus on the BERT model. BERT ([3]) means Bidirectional Encoder Representations from Transformers. BERT use transformer architecture to learn the text representations. There are pre-trained versions of BERT that can be already fine-tuned and used to solve your specific supervised learning task.

Table 2: Summary Statistics of Performance Among Neural Network Models

Model	Loss	Accuracy	Learning Rate	Epoch	Trainable Parameters	Model Size
FCN	0.94	59.02%	0.01000	50	2,863,619	11 MB
FCN with Embedding	1.52	63.92%	0.01000	40	103,861,563	415 MB
Conv1d with Embedding	3.37	66.24%	0.01000	100	1,095,739	4 MB
BERT	0.68	83.76%	0.00002	10	109,484,547	438 MB

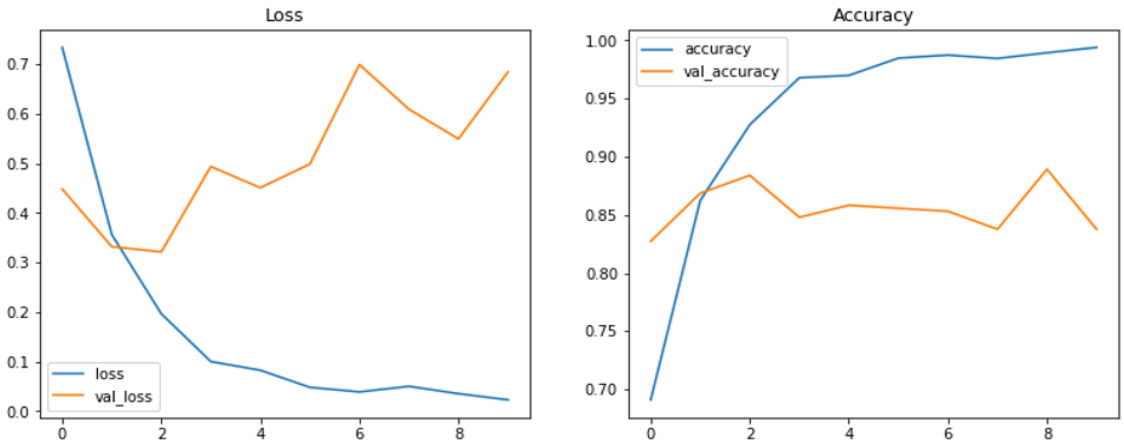


Figure 14: Performance of BERT Model

4.4 FinBERT Model

1. We firstly investigate the FinBERT which is a strong financial sentiment analysis tool with BERT (`https://github.com/ProsusAI/finBERT`). We get the dataset from FinancialPhraseBank and parse them into training, validation and test datasets. Using the datasets, we further fine-tune the FinBERT model.

2. Since FinBERT can only handle English, in order to get the sentimental score, we firstly translate our Chinese text in batch using googletrans API. Then we configure and fine-tune the FinBERT model, and use that to compute sentimental scores.

4.5 Fine-tune Chinese BERT Model

Considering the possible loss of accuracy from translating Chinese into English, we are motivated to fine-tune Chinese BERT Model in handling the sentimental analysis of equity research reports.

4.5.1 Data Preprocessing

1. Due to the lack of high-quality Chinese financial sentimental data with labels in sentences or short text, we decide to construct our own data for fine-tuning Chinese BERT Model. We combine a set of methods to avoid selection bias. For instance, we manually collect some

news and comments from Wind Platform, which is a financial database similar to Bloomberg but in Chinese. We also conduct web crawler to get some short comments and financial news from Sina Weibo (`https://github.com/fip-lab/Sentiment_Analysis`). Since some of those comments are from non-professionals, we need to manually select the part with high-quality. Then, we manually label the negative information with label -1, neutral information with label 0 and positive information with label 1. Moreover, we label the sentiment from the perspective of equity reports. For instance, news about the worsening condition of the COVID-19 is negative for the development of a country. However, for some stocks, like Zoom, they might benefit from these kinds of news. So, we would classify these kinds of news into "neutral" instead of "negative". We believe this dataset would be a great reference for further research on Chinese equity reports. Our final label file contains 1440 rows containing news, comments and analysis, which can be found in `https://github.com/PatternDetection/Project/blob/main/data/label/label_final.csv`.

2. Then, we use the Chinese financial sentimental data with labels we constructed before as the data for fine-tuning Chinese BERT Model. We firstly partition 80% of the data into training data and the remaining 20% of the data into validation data.

3. We then implement BertTokenizer to tokenize the input text. In the BertTokenizer, we use the directory containing Chinese BERT model as the vocab_file argument. Also, we set do_lower_case = True because we want to lowercase the input when tokenizing. We then use tokenizer.encode_plus() to tokenize and prepare for the model a pair of sequences. The output tokens from our tokenizer.encode_plus() is a dictionary with the keys "input_ids", "token_type_ids", "attention_mask".

4. Since we are doing classification on multiple classes, we then convert our labels to binary class matrix through one-hot encoding. Then, we create tf.data.Dataset using the tokenized results and implementing tf.data.Dataset.from_tensor_slices() function. We set the batch size to 32, and set train and validation shuffle buffer size equal to the number of data points in train and validation sets respectively. Finally, we create tf.data pipelines for the training and validation dataset through following the order - shuffle, batch, prefetch.

4.5.2 BERT Model Construction

1. TensorFlow models can be instantiated with from_pretrained() to load the weights of the encoder from a pretrained model. To conduct the classification task, we use TFBertForSequence-Classification.from_pretrained("bert-base-chinese", num_labels=3) to load the pre-trained weights of Chinese BERT model.

2. Then, we set learning_rate = 2e-5, epochs = 30, the optimizer to be Adam and the loss to be CategoricalCrossentropy because there are there are two more label classes. And we fine-tune our model using these parameters.

3. After that, we used the fine-tuned model to conduct prediction. Considering that BERT has a max length limit of tokens = 512, and some of our text information is super long, we first break all the text information into sentences. Motivated by FinBERT package, we process our predictions to be represented in three different columns: 1) logit: probabilities for classifying into each class, 2) prediction: predicted label, 3) sentiment_score: sentimental score

of each sentence is calculated as the probability of classifying into +1 minus the probability of classifying into -1. And the sentimental score of the text information is calculated by taking the average of the sentimental scores among all the sentences.

4.5.3 BERT Model Performance

Figure 15 shows how the loss and accuracy in the training and validation datasets evolve during the training process. We end up with an accuracy of 0.7535 in validation dataset after fine-tuning Chinese BERT model.

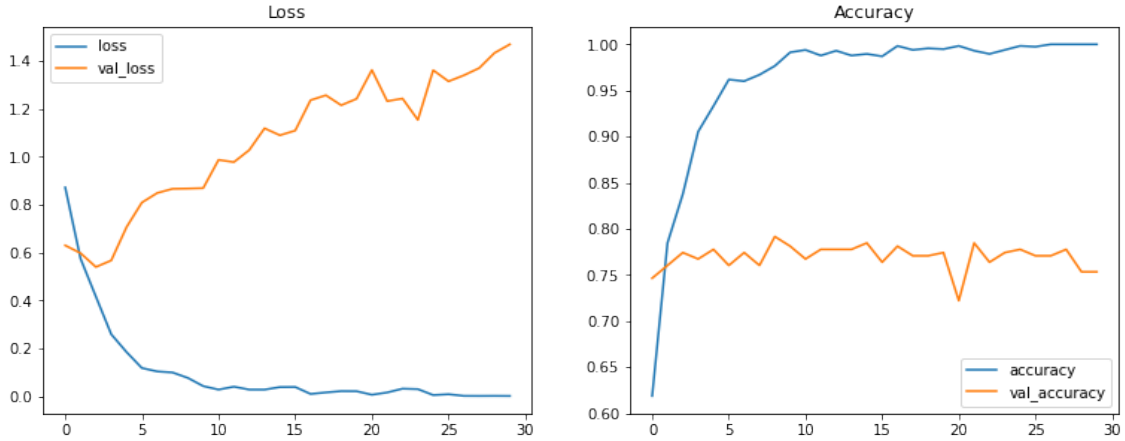


Figure 15: Performance of Chinese BERT Model

4.5.4 Result Comparison

Since we use similar model structure and methods to define the sentimental score, we then compare the sentimental score calculated from FinBERT model and our fine-tuned Chinese BERT model. Figure 16 shows the comparison result. Although the exact values for the sentimental score varies a lot, we can see the general trend is positively correlated. When English FinBERT model gives positive sentimental score, our fine-tuned Chinese BERT model rarely gives negative sentimental score. And the correlation between these two sentimental scores over our 628 text files is 0.399. We could observe some consistency between the sentimental score calculated from FinBERT model and our fine-tuned Chinese BERT model.

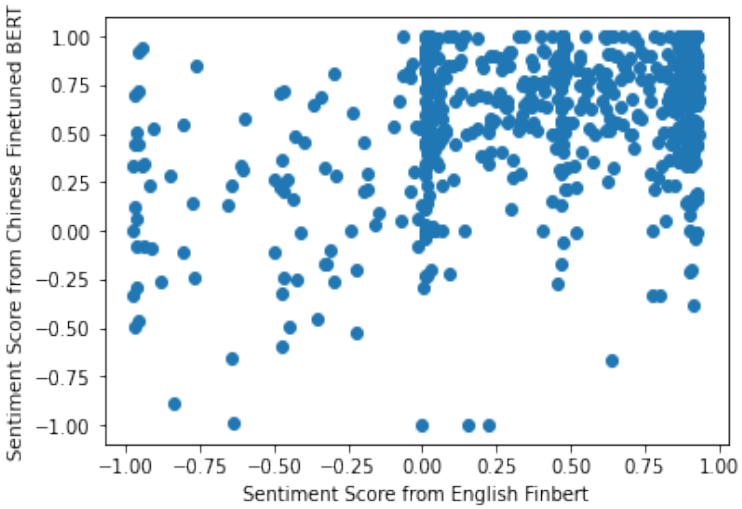


Figure 16: Sentimental Score Comparison

5 Analysis on Sentimental Score

To analyze the predictability of our sentimental factor, we firstly calculate the 1-trading-day, 5-trading-day, 10-trading-day, 20-trading-day, 30-trading-day, 60-trading-day and 100-trading-day return after the date the equity report is posted. Then, we calculate the correlation between our sentimental score and those returns. The following result show that the correlation between sentimental score and returns is maximized during 20 to 60 trading days, which is in accordance with our expectation. Different from quick response of stock price to instant financial news, equity reports tend to give midterm forecasts by taking the discounted cash flow in the future into analysis.

Table 3: Correlation Between Sentimental Score and Returns

Return	Correlation
1-trading-day return	0.010871
5-trading-day return	-0.016624
10-trading-day return	0.010887
20-trading-day return	0.073622
30-trading-day return	0.072292
60-trading-day return	0.073875
100-trading-day return	0.000638

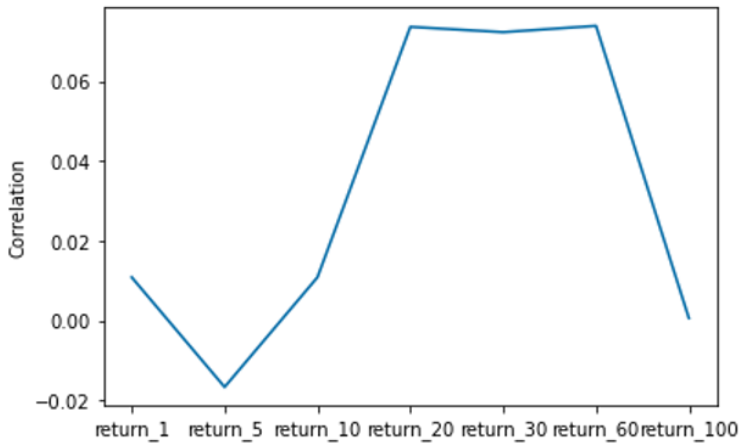


Figure 17: Relationship Between Correlation and Return with Different Frequency

Considering the difficulty of predicting the equity market, we regard our correlation to be fairly reasonable. We also compare the values with some fundamental factors and indicators widely used in investment, like P_EBITDA and PE. Table 4 shows the correlation between fundamental factors and returns we obtain from an equity report from Research Department, GuangDa Securities in China. Compared with those widely used fundamental factors, our sentimental scores show stronger correlation and better predictability although the values of our correlation are not large due to the difficulty of predicting the market.

Table 4: Correlation Between Fundamental Factors and Returns

Fundamental Factor	Correlation
PE	0.0249
EP	0.0179
P_EBITDA	0.0314
PCF	0.0393
PEG_3Y	-0.0015
DP	0.0244

Also, from the visualization plot of some randomly chosen points we can see there exist some clear points with high correlation in movements between sentimental score and returns.

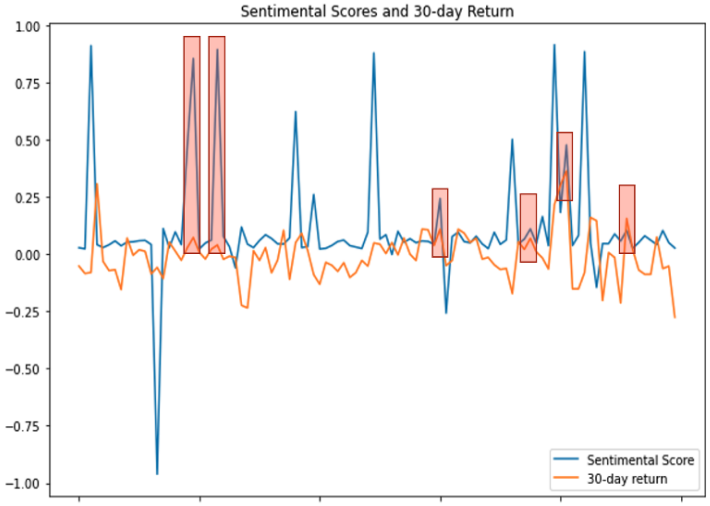


Figure 18: Relationship Between Sentimental Score and 30-trading-day Return

We also conduct a back-test in Chinese equity markets using our sentimental factors from 2020 to 2021, since we have sufficient data points in these two years. We also compare with some common factors in Chinese equity markets. We observe that our sentimental factor achieve an annual Sharpe Ratio of 5.2 while the 30-day-return factor, which is constructed using the moving average of the previous return over 30 days, gain an annual Sharpe Ratio of -0.61. Compared with the correlation with the market return, the back-testing gives us more insights on the excessive return. This also reveals that the sentimental factors have not been widely used in investment in Chinese equity market. There exists strong potential in the investment opportunities of our sentimental factors.

6 Next Step

As for the OCR part in CV, because we use Baidu OCR, we are facing the limits of calls per day. Therefore we implement the type without specifying location of the text. To achieve better precision, we shall discover more tools or manually make modification to improve some public models.

As for NLP, we will try to construct a high-quality dataset with more data points. Now the labels are manually selected from over 10,000 records, so the human errors might play a role here. The next step will be the improvement of the dataset, which would greatly improve the accuracy and efficiency of fine-tuning our Chinese BERT model.

References

[1] Apostolos Antonacopoulos et al. "A Realistic Dataset for Performance Evaluation of Document Layout Analysis". In: *2009 10th International Conference on Document Analysis and Recognition*. 2009, pp. 296–300. DOI: 10.1109/ICDAR.2009.271.

[2] J. Cartucho, R. Ventura, and M. Veloso. "Robust Object Recognition Through Symbiotic Deep Learning In Mobile Robots". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 2336–2341.

[3] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[4] *GreenKey Releases NLP Tool Enables Traders to Process Sell-Side Research in Minutes*. eng. Coventry, 2020.

[5] Benjamin Charles Germain Lee et al. "The Newspaper Navigator Dataset: Extracting Headlines and Visual Content from 16 Million Historic Newspaper Pages in Chronicling America". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. CIKM '20. Virtual Event, Ireland: Association for Computing Machinery, 2020, pp. 3055–3062. ISBN: 9781450368599. DOI: 10.1145/3340531.3412767. URL: https://doi.org/10.1145/3340531.3412767.

[6] Anirban Mondal and Atul Singh. "Emerging Technologies and Opportunities for Innovation in Financial Data Analytics: A Perspective". eng. In: *Big Data Analytics*. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 126–136. ISBN: 9783030047795.

[7] Zejiang Shen, Kaixuan Zhang, and Melissa Dell. "A Large Dataset of Historical Japanese Documents with Complex Layouts". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (June 2020). DOI: 10.1109/cvprw50498.2020.00282. URL: http://dx.doi.org/10.1109/CVPRW50498.2020.00282.

[8] Zejiang Shen et al. "LayoutParser: A Unified Toolkit for Deep Learning Based Document Image Analysis". In: *arXiv preprint arXiv:2103.15348* (2021).

[9] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[10] Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. "PubLayNet: largest dataset ever for document layout analysis". In: *2019 International Conference on Document Analysis and Recognition (ICDAR)*. IEEE. Sept. 2019, pp. 1015–1022. DOI: 10.1109/ICDAR.2019.00166.