Jiabo/Roberta-Chinese sentiment: a Binary (neg/pos) Sentiment Analysis of Twitter Posts about The Taipei Mayoral Elections with BERT

Jasper Hewitt 嘉博

National Chengchi University

Data science

Professor: Owen Lu

Introduction and Problem statement

Students in the ICI capstone project were looking for a way to do sentiment analysis about the Taiwanese local elections on Weibo. They decided to use BERT and came to Owen and me for help. Owen and I have been looking for ways to properly fine-tune a Chinese Bert model, but the current accuracy is still low. Partly because we have only limited labelled training data. Therefore, I want to do more research into how we can best fine-tune this BERT model in order to push up the accuracy rate. I have already started playing around with ChatGPT to see if I can use the sentences it creates to improve the BERT model. The ultimate goal of this project is to create a large table that explores the accuracy of several different training strategies. Subsequently, I want to use this model to look at Twitter instead of Weibo. Given that the ICI capstone project is already looking at the sentiment on Weibo, I thought it would be interesting to do a sentiment analysis on Twitter. This also comes with another challenge: I will have to find a Twitter scraper myself to harvest the data I need.

Research question

How to best fine-tune a Chinese BERT model to do sentiment analysis about the Taiwanese local elections on Twitter?

- create a table with the accuracy of different training possibilities to find the most accurate option.
- Use this accurate model to do the sentiment analysis on Twitter.

Literature review

Ever since the original BERT article was released in 2019 (Devlin et al.), researchers immediately started to redevelop their own models from scratch based on domain specific language. Examples include BioBert (Lee et al., 2019), BERTweet (Nguyen et al., 2020), SCIBERT (Beltagy et al., 2019), and LEGAL-BERT (Chalkidis et al, 2020). All of the aforementioned models outperformed BERT-BASE on several NLP tasks. Amongst these models, BERTweet is especially relevant for social media sentiment analysis. However, there is one problem: BERTweet does not read Chinese. Luckily, there is a GitHub repository that lists all of the Chinese models that have been created over the last few years. This means that we will not have to start pre-training our own model from scratch, but we will only have to fine-tune it for our specific sentiment analysis. There are several public databases with Chinese pre-labelled data for sentiment analysis that we can use to fine-tune our model. For example, there is the Weibo challenge dataset that we received through the ICI project, and another dataset on GitHub. For this project, we will play around with different combinations of pre-trained models and fine-tuning data to see what yields the best results. Some of the models that we will test include 'hfl/chinese-roberta-wwm-ext' and 'hfl/chinese-bert-wwm'.

Methodology

There are several steps in our process:

- Step 1: pick a model.
    - There are several different Chinese BERT models, choosing the right one will be very important for our accuracy. We will test the accuracy of different models with our training and testing data to see which one yields the highest results. However, given Roberta's dominance over the past few years, it is likely going to be this one.

- Step 2: fine-tune the model
    - 2.1: find a solution for our unbalanced training data.
        - Our training data is very unbalanced. It contains significantly more negative than positive posts. We will explore several options for dealing with this problem like finding additional positive sentences from other datasets or ChatGPT. In addition, we will also explore some other popular methods to deal with unbalanced data, like adding class weights, upsampling, and downsampling.

- Step 3: data collection.
    - We will look for a Twitter scraper that can directly scrape all of the posts with a specific search term. This is where all of the data collection of this project happens.

- Step 4: data visualization
    - In this part we will plot the data to visualize our results. We will create a bar plot that shows the total number of posts per candidate, pie charts that show the percentage of positive and negative posts per candidate, and a day-by-day overview of the positive and negative posts per candidate. This later will also include a line plot that shows the percentage of positive posts on that specific day.

2.1 picking a model

Ymcui's Github page provides a wide range of different Chinese BERT models. We initially started out with their basic hfl/chinese-bert-wwm. However, after some additional research we came to the conclusion that other models would probably perform better. The table underneath provides a comparison of two models based on our basic training set and testing data. More information on the training and testing data will be provided in the next chapter. We eventually decided to go with hfl/chinese-roberta-wmm-ext because it gave us the best results. This is in line with the general expectation that roberta performs better than basic Bert. It is important to note that hfl also provides a bigger option: hfl/chinese-roberta-wwm-ext-large, but we are unable to run this in our Colab environment because of limited memory space.

Model	Test accuracy
hfl/chinese-bert-wwm	Test: 0.8789
hfl/chinese-roberta-wwm-ext	Test: 0. 8878

2.2 Fine-tuning

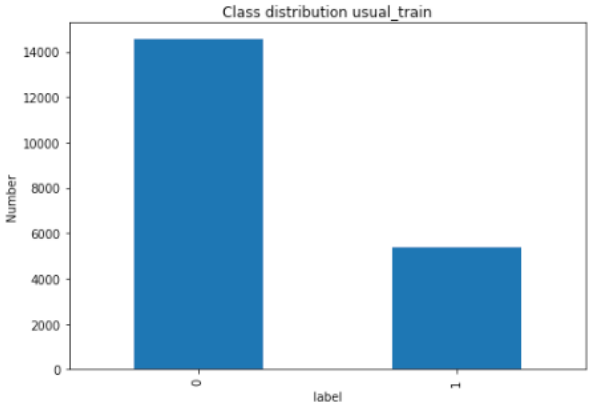
To fine-tune bert we used the sample code provided by Hugging Face in combination with other tutorial videos, Stack Overflow, and ChatGPT.

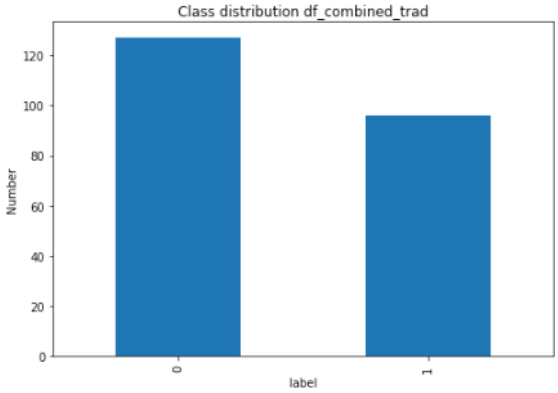
2.1 training, evaluation, and testing data

The ICI professor who asked us for help had already provided us with some data that we could use. For training data, he referred us to the 'The Evaluation of Weibo Emotion Classification Technology, 'SMP2020-EWECT', (in Chinese: SMP2020微博情绪分类技术评测). This dataset contained our training and evaluation data. In addition, he provided us with a testing dataset that contained Weibo posts about Chiang Wan-An (sample_combined). This dataset was already manually labelled by ICI students. We consider this testing data to be the most important measure, because the language will be similar to our target language than the basic evaluation dataset from SMP2020-EWECT. However, there was one big problem with the testing data: it was very dirty. We used an elaborate regex to get rid of all the titles, links and other noise (see notebook). An overview of the training, evaluation, and testing data can be found in the table below. All the files that we used can be found in the repository.

name	Total	Positive/negative
usual_train	27768	Negative: 14553 Positive: 5378
usual_eval	2000	Negative:1019 Positive:391
df_combined_trad	399	Negative: 127 Positive: 96

The training data from the SMP2020-EWECT contains clean one/two sentence posts that were taken from *Weibo*. All the files on the right were originally divided in six categories. Angry, fear, sad, neutral, happy, and surprise. We initially planned to fine-tune our model with these 6 categories. However, the accuracy was so low that we decided to just go for two categories: positive and negative. In order to convert labels to positive and negative, we decided to convert ‘angry’, ‘fear’ and ‘sad’ into ‘negative’, and ‘happy’ into ‘positive’. This obviously led to an unbalanced dataset. There are significantly more negative than positive posts, which may reduce our model’s ability to detect positive posts in the future. Therefore, we had to think of ways to balance out the data. The evaluation data is also unbalanced. However, because this one is not very important but serves more as a reference, we decided to leave that one as it was. On the other hand, the more important testing dataset is already relatively balanced, so no adjustments were needed. The training and testing data is visualized in the table below.





2.2 Solving the unbalanced training data problem

Step 1: finding additional positive data

The first thing we tried was to find additional positive data. There were several places where we found this data. Firstly, the SMP2020-EWECT data actually contained a few additional datasets that we were initially not using in our project: virus_train and usual_test. Usual_test was the original testing dataset of SMP2020-EWECT which we decided not to use because we have our own testing dataset. The virus_train dataset contains posts that are specifically about covid. In addition, I found another dataset with positive and negative hotel reviews on Github. Finally, we consulted Open AI's ChatGPT to create some additional positive sentences on politicians. ChatGPT can be a bit tricky and it is difficult to extract large amounts of data from it. I managed to do this by creating a while loop that requests it to create sentences multiple times (see notebook). Because ChatGPT tends to become repetitive very quickly, we set the frequency_penalty to 0.2 and the presence_penalty to 0.4. This forces ChatGPT to be less repetitive. We eventually got 520 sentences from ChatGPT. It would have been possible to get more sentences if we decided to reduce the penalties, but the data would have been too bad.

Name	Total	Positive/negative
Virus_train	8606	Negative: 2526 Positive: 4423
Usual_test	5000	Negative:2618 Positive: 1018
GPT_ positive_ sentences	520	Positive: 520
Review_pos	9146	Negative: 4561 Positive:4581

We fine-tuned hfl/chinese-roberta-wwm-ext with different combinations of our testing data to see what yields the best response. A large table with an overview, and links to the respective notebooks, can be found on the next page. As can be seen in the table, taking the additional positive data from the virus_train and review_pos datasets actually leads to a heavy loss in accuracy. A possible explanation for this is that the data is simply too specific and different from usual_train and the testing dataset. Similarly, adding the positive sentences from ChatGPT also reduces the accuracy of the model. This shows that the sentences produced by ChatGPT are still too repetitive and likely cause the model to overfit. The best composition of training data is usual_train plus the positive posts of usual_test, trained on only 1 epoch.

Table for the right combination of training data

Model	Training data	1 epoch	2 epochs	3 epochs	4 epochs
Hfl/Chinese-Roberta-wmm-ext	Usual_train	Eval: 0.9297 Test: 0. 8878	Eval:0.9327 Test:0.8565	Eval:0.9255 Test:0.8834	Eval:0.9170 Test:0.8699
	Usual_train Usual_test (pos) Review_train (pos) Virus_train (pos)	Eval:0.9219 Test:0.8430	Eval: 0.9234 Test:0.8385	Eval:0.9191 Test:0.7130	
	Usual_train Usual_test	Eval:0.9269 Test:0.8923	Eval:0.9212 Test:0.8923	Eval: 0.9255 Test:0.8968	
	Usual_train Usual_test (pos)	Eval:0.9212 Test:0.9058	Eval:0.9198 Test:0.9013	Eval:9212 Test: 0.8923	
	Usual_train Usual_test(pos) GPT_(pos)	Eval:0.9205 Test:0.9013			
hfl/Chinese-bert-wmm	Usual_train	Eval: 0.9141 Test: 0.8789	Eval: 0.9191 Test: 0.8878	Eval: 0.9156 Test:0.8475	

Step 2: upsampling, downsampling, and adding class weights

Now that we found the best combination of data (usual_train plus the positive posts from usual_test, total) it is time to explore different methods to fix our unbalanced data problem. We explored several options like upsampling, downsampling, and adding class weights. Firstly, upsampling and downsampling is logical method do deal with unbalanced data. I used the resample function from the sklearn.utils package. However, both upsampling, downsampling, and a combination of both led to worse results. This is probably because the upsample function basically just copies some of the positive posts, which results in a lot of duplicates and an increased chance of overfitting. As for downsampling, this just leads to a very small training dataset. It turns out that Adjusting class weight gives us the best results. By adding a certain weight to classes, you are basically telling Bert to focus a little more on a certain class, while focussing a little less on another. I found this solution on the Hugging Face website (Hugging Face, n.d). Hugging Face also provides instructions for how to assign class weight to Bert's trainer. The only thing that I still had to do was calculating the class weight. I did this with a method I got from ChatGPT.

```
Label_counts = training_pd['label'].value_counts()
weights = (len(training_pd) / label_counts)
```

The class weights are 'negative' = 1.457642, and 'positive' = 3.185115. This is the same result as the multiplied output of the class_weight function from sklearn.utils package.

Method	1 epoch	2 epochs	3 epochs
Adjusting class weight	Eval: 0.9304 Test:0.9058	Eval: 0.9219 Test:0.8923	Eval:0.9226 Test:0.8789
upsampling	Eval: 0.9241 Test:0.8923	Eval:0.9241 Test:0.8968	Eval:0.9134 Test:0.8699
Downsampling	Eval: 0.9021 Test:0.8161	Eval:0.9134 Test:0.8699	Eval:0.9042 Test:0.8744
Up+downsampling	Eval:0.9234 Test:0.8744	Eval: 0.9319 Test:0.8699	eval:9212 Test:0.8834

Despite the fact that adjusting the class weight did not lead to a higher accuracy on the testing dataset, it did significantly improve the score on the evaluation dataset. I have therefore decided to make this one the final model that I will use for predicting the sentiment on twitter.

2.3 pushing model to Huggingface

In order to store this model, I pushed the model to Hugging Face. Storing a model in Huggingface has several benefits. For example, when we want to use the fine-tuned model for an actual prediction task, we no longer have re-finetune it before doing the prediction task. This saves a lot of time, energy, and GPU space. In addition, now that the model is stored, anyone who wishes to use it is able to import it straight into their developing environment.

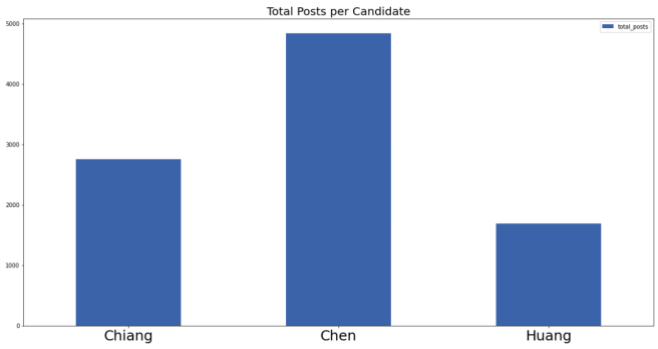
In order to store a model in Huggingface you first need to make an account, then you have to create a model card, and then you have to create an access key that allows you to 'write' something on the model. I stored my fine-tuned model as Jiabo-Roberta-Chinese-sentiment.

3 Analysis

3.1 data collection (tweet_scraper) (9285 posts)

Now that the Bert model is properly fine-tuned, it is time to get to get to the actual analysis. The first step will be to gather some data. I am aware that Twitter and other websites offer APIs. However, these APIs are limited and you often have to pay. Therefore, I decided to find a way to scrape tweets on my own. I ended up on the following Github repository and corresponding YouTube tutorial by Izzy Analytics. I had to make a few adjustments because Twitter has recently made some changes to their login page and I used Chrome for Mac instead of Microsoft Edge for Windows (see our notebook). The scraper uses Selenium and a web driver for Chrome to scrape all the tweets for a certain search term. If you want to use the Twitter scraper yourself you will first have to download Selenium and a Chrome driver. Because the code will have to run these local programs, I suggest you run it in a local Jupyter Notebook. In total, I collected 9285 posts about the candidates. The table below shows the number of posts per candidate. The excel sheets can be found in the repository. Just like with the testing dataset with the posts from Weibo, the scraped tweets were very dirty so the final notebook contains a very elaborate regex to delete all the noise like usernames, dates, and tags such as 'Replying to'. In addition, because Chen Shih-Chung is the only candidate who is using Twitter, we also deleted his own posts about himself from the data. Finally, the scraped tweets are in traditional Chinese, whereas the model is trained on simplified Chinese. We tested the model's performance on the dataset with traditional characters and this actually resulted in a slight drop in accuracy. If you import the model and directly test the simplified Chinese testing data, it scores 0.9108 on the simplified dataset,[1] while only scoring 0.8968 on the traditional dataset. Therefore, we decided to convert the tweets from traditional to simplified characters with the openCC package (see notebook).

Candidate	posts	dates
Huang Shan-shan (黃珊珊)	1688	10/31-12/09
Chiang Wan-an (蔣萬安)	2755	11/01 – 12/10
Chen Shih-chung	4842	11/01 – 12/10

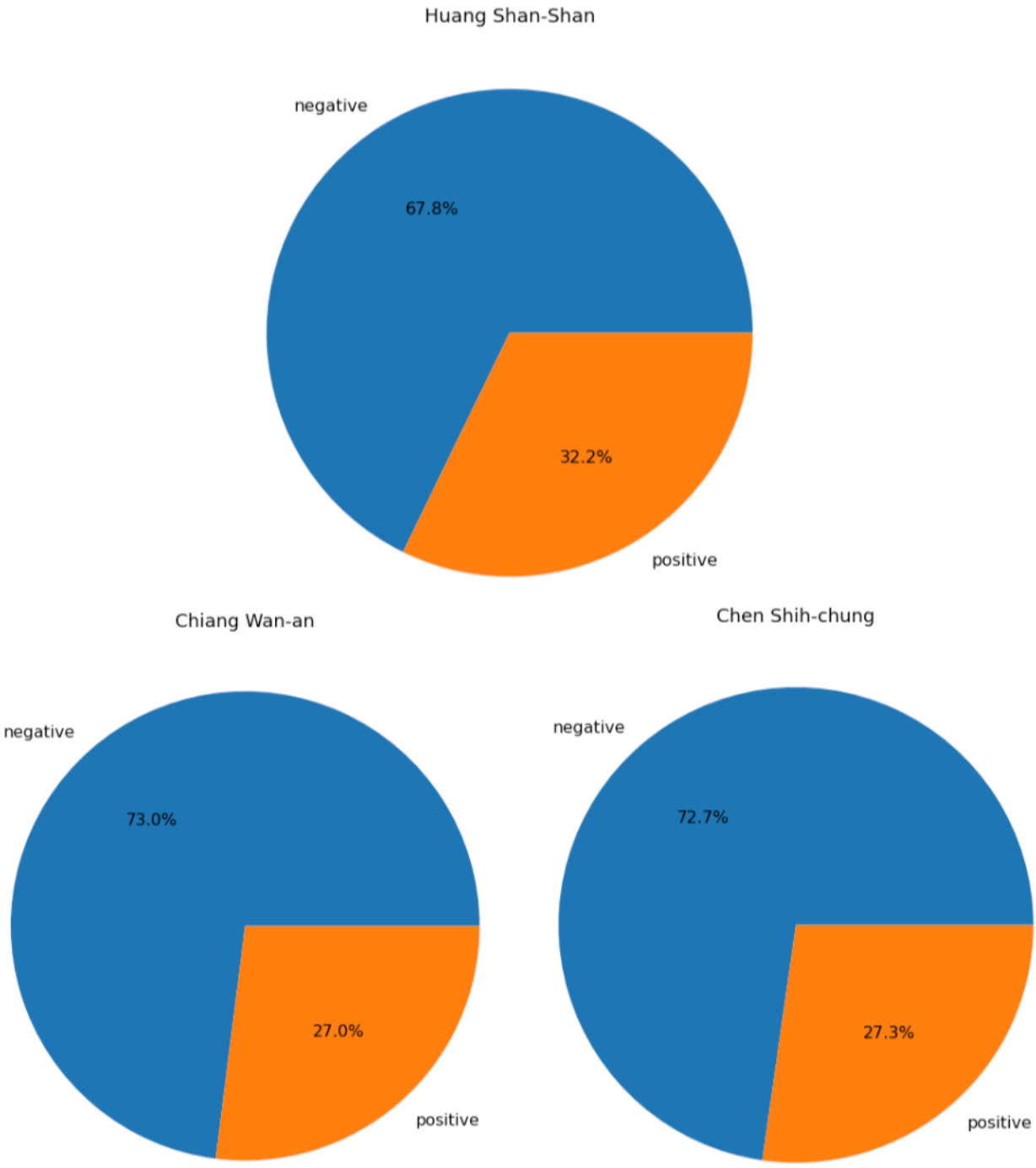


[1] This is a very puzzling phenomenon. In the training notebook the model scores only a 0.9058 on the testing dataset. However, when you directly import the fine-tuned model from Hugging Face and test it on the same dataset it suddenly scores 0.9108. I believe this is no coincidence because it happened many times before when a was training our other models on different testing datasets. I still do not know why this happens.

3.2 results and data visualization

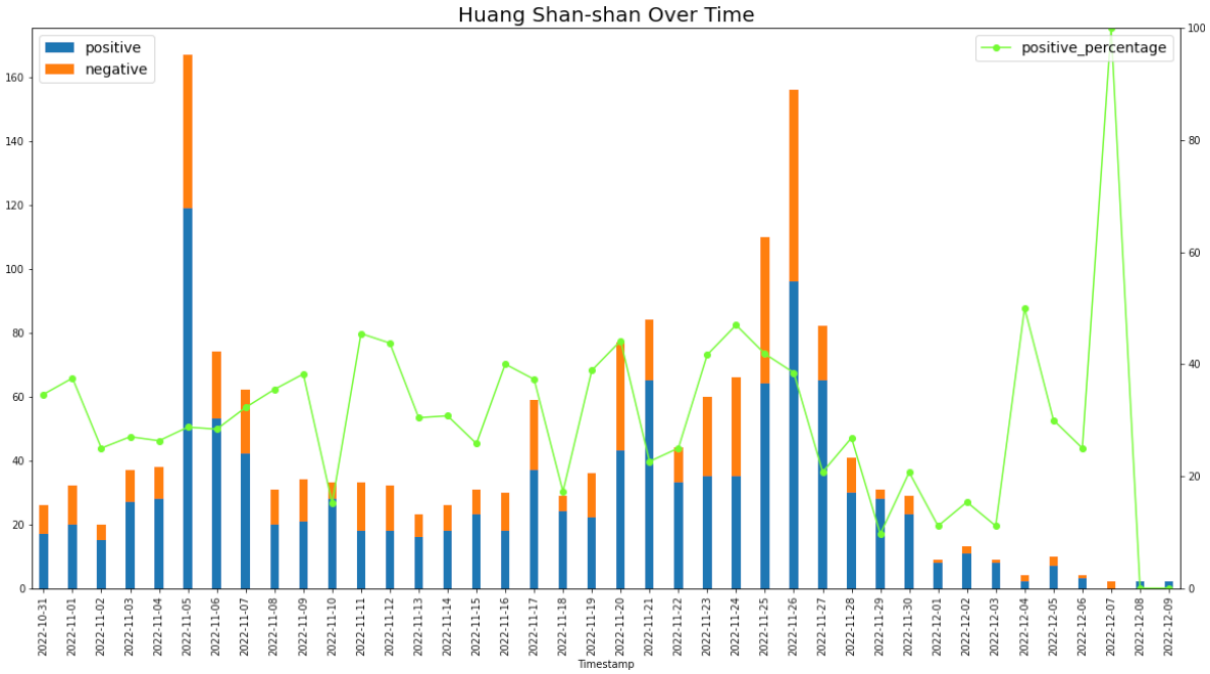
Pie charts

I think pie charts provide a nice overview to show the percentage of positive and negative posts per candidate. It is interesting to see that Huang Shan-shan enjoyed the highest percentage of positive posts, despite getting the lowest number of votes in the actual election. However, since Twitter is not a very widely used platform in Taiwan, it makes sense that the sentiment on Twitter does not represent the actual outcome of the elections. Therefore, this research really only tells us something about the sentiment surrounding these candidates *on Twitter*. The findings cannot be generalized.

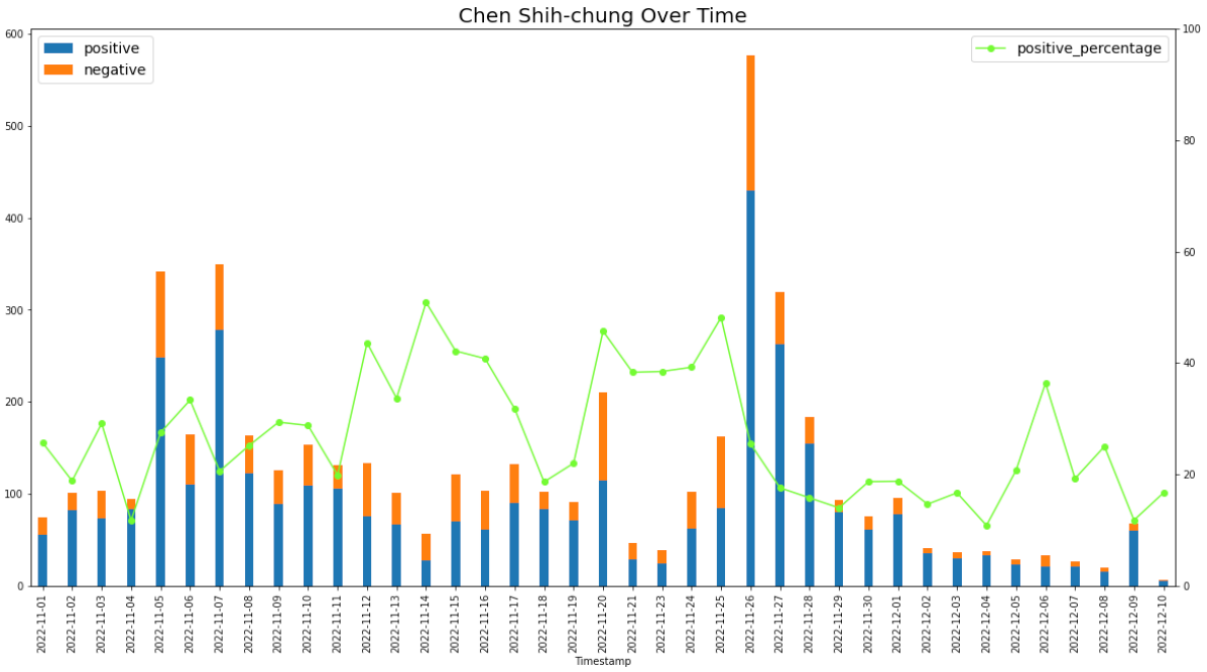


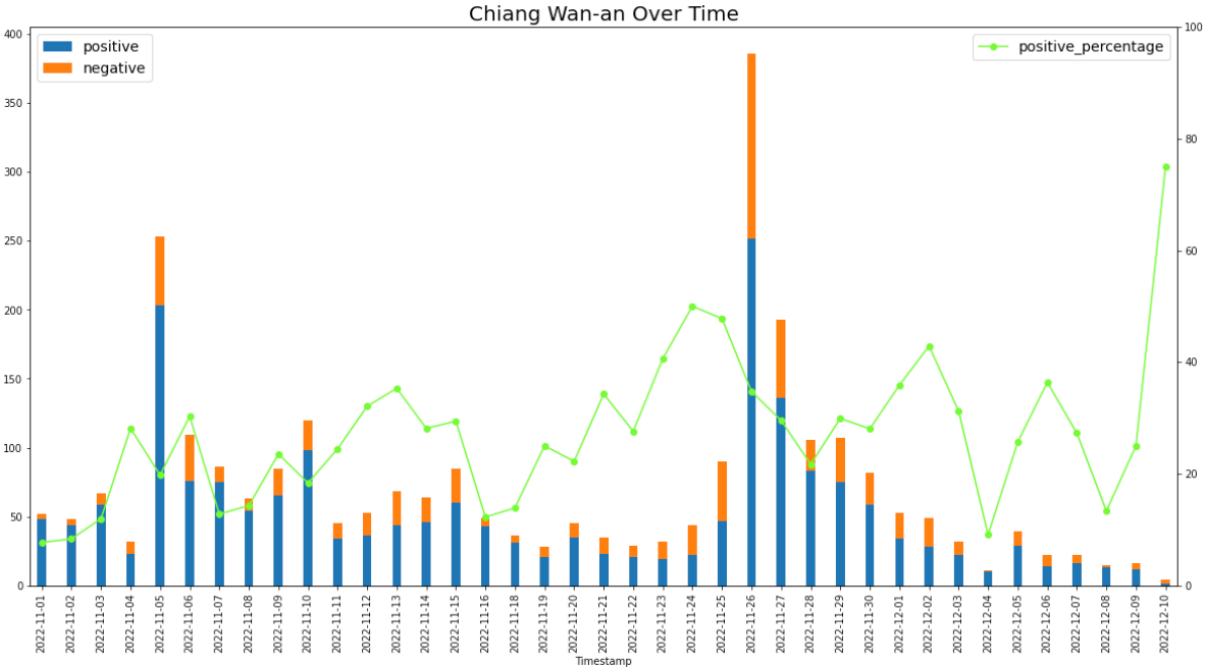
Day by day overview

The next few plots show a more in-depth analysis about the Twitter sentiment over time. The bars represent the number of positive (blue) and negative (orange) posts per day; the numbers are represented on the left Y axis. In addition, the green line and the right Y axis show the percentage of positive posts relative to that specific day. We can see an uptick in activity for all candidates on 05/11 (tv debate) and 26/11 (elections). Another interesting finding is that all three candidates have a three-day period right before the elections where the number of posts *and* the percentage of positive posts increased. The percentage of positive posts fell again on election day, even for Chiang Wan-an, despite him winning the election. After the elections, posts about the candidates drastically reduced to not even a dozen per day.



(See next page for Chiang Wan-an and Chen Shih-chung)


Chen Shih-chung Over Time
positive
negative
positive_percentage
Timestamp


Chiang Wan-an Over Time
positive
negative
positive_percentage
Timestamp

4 Conclusion

In this paper we answered the following research question: How to best fine-tune a Chinese BERT model to do a binary sentiment analysis about the Taiwanese local elections on Twitter? In order to answer this question, we first compared the hfl/chinese-roberta-wwm-ext' and 'hfl/chinese-bert-wwm' and concluded that the latter was the better model. we then explored several ways to deal with our unbalanced training data and found that a combination of adding positive sentences and adding class weights yields the best results (Test:0.9058). Subsequently, we pushed the model, Jiabo-Roberta-Chinese-sentiment, to Huggingface. We then adjusted a Twitter scraper and scraped 9285 tweets, let our model predict the sentiment, and visualized the data (see notebook). We converted the traditional characters in the tweets to simplified, since the model performs worse with traditional characters.

As with any research, there are a few shortcomings that can be readdressed in future projects. Firstly, a binary model like Jiabo-Roberta-Chinese-sentiment may be misleading because it will classify all posts as either positive or negative, also the neutral ones. Future research can try to find, or manually label, additional training data to make a three-labelled model with reasonable accuracy. Furthermore, it is likely that the model hfl/chinese-roberta-wwm-ext-large would yield even better results after fine-tuning with our data. Unfortunately, I was not able to fine-tune this model because of limitations in Colab. If you have access to a more powerful environment this would be an interesting option to explore.

On the other hand, there are also two interesting things that we can learn from this paper. Firstly, we found that adjusting class weights was the best way to deal with unbalanced data in this case. In addition, we also found that ChatGPT is not strong enough to produce training data for BERT model. The sentences simply become too repetitive.

Cited works

1. Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. 2020. "BERTweet: A pre-trained language model for EnglishTweets." In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 9–14, Online. Association for Computational Linguistics.

2. Hugging Face. N.d. "fine-tune a pretrained model." https://huggingface.co/docs/transformers/training

3. Hugging Face. N.d. "trainer." https://huggingface.co/docs/transformers/main_classes/trainer.

4. Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. "SciBERT: A pretrained language model for scientific text." In *Proceedings of EMNLP-IJCNLP*, pages 3615– 3620.

5. Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. 2020. "LEGAL-BERT: The Muppets straight out of Law School." In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2898–2904, Online. Association for Computational Linguistics.

6. Israel-Dryer. 2021. "Twitter-Scraper." Github Repository. https://github.com/israel-dryer/Twitter-Scraper/blob/main/twitter-scraper-tut.ipynb.

7. Izzy Analytics. 2020. "Twitter scraper Python tutorial." *Youtube*. https://www.youtube.com/watch?v=3KaffTIZ5II&t=1063s.

8. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. "BERT: Pre-training of deep bidirectional transformers for language under- standing." In *Proceedings of NAACL*, pages 4171– 4186.

9. Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2019. "BioBERT: a pre-trained biomedical language representation model for biomedical text mining." *Bioinformatics*, page btz682.

10. SMP2020微博情绪分类技术评测（SMP2020-EWECT）
https://smp2020ewect.github.io/.

11. Ymcui. 2022. "Chinese-Bert-wmm." Github Repository. https://github.com/ymcui/Chinese-BERT-wwm.