UNITED INTERNATIONAL COLLEGE

Data Mining (For DS students) (1002)

Dr. Zongwei LUO

Introduction and Implementation of Sentiment Analysis Techniques (based on Weibo data) and its simple application: Sentiment Evolution Analysis of Weibo during COVID-19

Group Project Report

Group 32

陆天宇	LU Tianyu	1930026092
董奂辰	DONG Huanchen	1930026024
赵一潘	ZHAO Yipan	1930026171

2021.12.19

Contents

Introduction

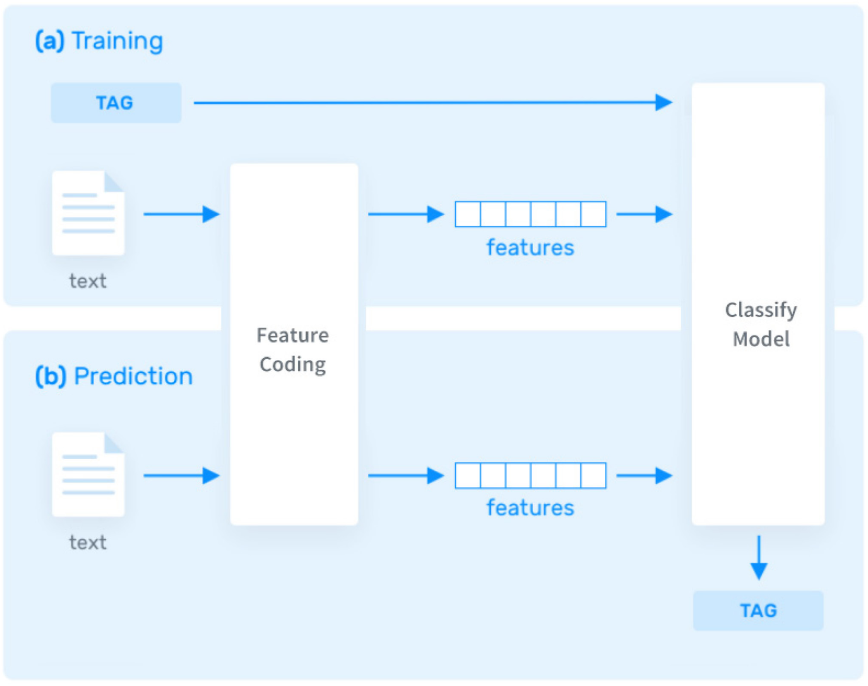
Sentiment analysis aims at identifying people’s emotion from a given sentence by using the extracted feaures, generally it is a job belongs to the nature language process area. Our research mainly focus on the binary classification of sentiment, in other words, the sentiment of a given sentence will be classified as either positive or negative. Although ternary classification of sentiment is much more rational, we did not do this as we cannot find suitable dataset. We shall do our sentiment analysis job on the data from weibo, which is a micro-blogging site commonly referred to as "Chinese Twitter", we choose it because its data has these characteristics: large data volume, emotion clarity. To be more specific, weibo’s financial report shows that it has over 200 million daily active users, which will result in a sufficient large data volume. According to online surveys and our daily experience, many people will express their sentiment directly and undisguised on weibo, although this might be annoying for common people as it caused many contradictions, the emotion clarity it resulted in is just suitable for our classifier training.

Our purpose of using sentiment analysis is to identify the public opinion and find its correlation with specific events, and this is shown to be helpful for government to make decisions from completed researches, for example, researchers from China People's Police University have already tried to apply this idea on "8-12" Tianjin Port Heavy Fire Explosion Accident and obtained a rational result.

1 Survey on Sentiment Analysis

1.1 Procedure of Sentiment Analysis

The main idea of sentiment analysis is to identify the emotion based on the features extracted. No matter which technique you choose, sentiment analysis generally follows a specific procedure, shown as the graph. Therefore, we need to train and use two models, one is for feature coding, another one is for classification.



1.2 Most Commonly Used Techniques for Sentiment Analysis

- Naïve Bayes & SVM

Pang, B., Lee, L., & Vaithyanathan, S. (2002, July). Thumbs up? Sentiment Classification using Machine Learning Techniques. *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, 79–86. doi:10.3115/1118693.1118704

- XGBoost

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. Presented at the San Francisco, California, USA. doi:10.1145/2939672.2939785

Ming, J. (2020, April 20). *Sentiment Classification using XGBoost*. Artificial Intelligence in Plain English. https://ai.plainenglish.io/sentiment-classification-using-xgboost-7abdaf4771f9

- BERT

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv [cs.CL]. http://arxiv.org/abs/1810.04805

Xu, H., Liu, B., Shu, L., & Yu, P. S. (2019). *BERT Post-Training for Review Reading Comprehension and Aspect-based Sentiment Analysis*. arXiv [cs.CL]. http://arxiv.org/abs/1904.02232

- LSTM

Sherstinsky, A. (2020). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena*, *404*, 132306. doi:10.1016/j.physd.2019.132306

Monika, R., Deivalakshmi, S., & Janet, B. (2019). Sentiment Analysis of US Airlines Tweets Using LSTM/RNN. *2019 IEEE 9th International Conference on Advanced Computing (IACC)*, 92–95. doi:10.1109/IACC48062.2019.8971592

- SnowNLP

Chen, C., Chen, J., & Shi, C. (2018). Research on Credit Evaluation Model of Online Store Based on SnowNLP. *E3S Web of Conferences*, *53*, 03039. EDP Sciences.

1.3 Most Commonly Used Techniques for Feature Coding

- Bag-of-Words (BOW)

Zhang, Y., Jin, R. & Zhou, ZH. Understanding bag-of-words model: a statistical framework. *Int. J. Mach. Learn. & Cyber.* **1,** 43–52 (2010). https://doi.org/10.1007/s13042-010-0001-0

- TF-IDF

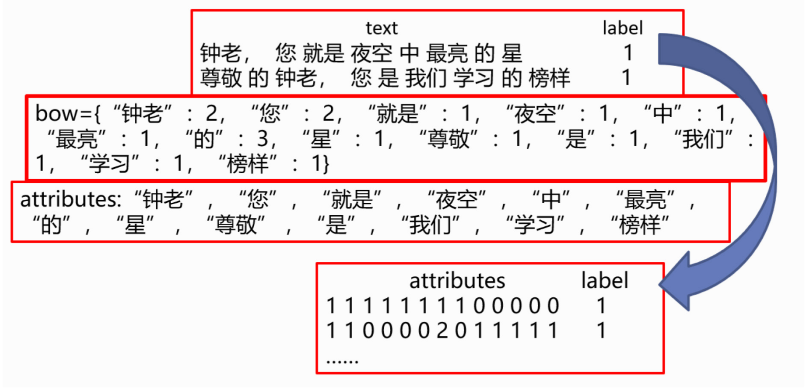
Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*. Vol. 28 No. 1, pp. 11-21. https://doi.org/10.1108/eb026526

2 Implemented Feature Coding Techniques

All the techniques for sentiment analysis are highly dependent to feature coding, therefore we use two feature coding techniques – Bag-of-Words and TF-IDF.

2.1 Bag-of-Word (BoW)

The Bag of Words model (BoW) is to put all Words into a Bag, regardless of the morphology and word order, that is, each word is independent, every word is counted, and the number of occurrences of each word is counted. In other words, the word bag model does not consider the contextual relationship between words in the text, but only considers the weight of all words, and the weight is related to the frequency of words in the text. For example:



text	label
钟老，　您 就是 夜空 中 最亮 的 星	1
尊敬 的 钟老，　您 是 我们 学习 的 榜样	1

```
bow={“钟老”：2，“您”：2，“就是”：1，“夜空”：1，“中”：1，
“最亮”：1，“的”：3，“星”：1，“尊敬”：1，“是”：1，“我们”：
1，“学习”：1，“榜样”：1}
```

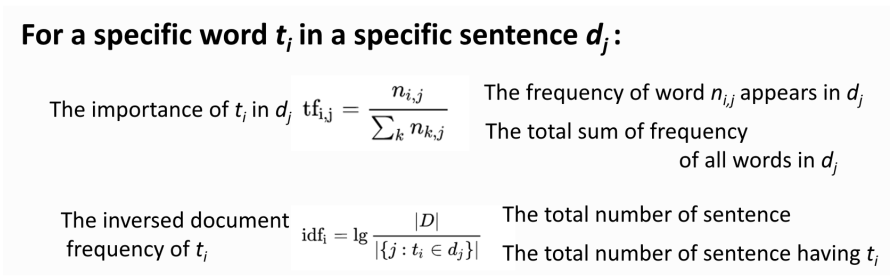
attributes:“钟老”，“您”，“就是”，“夜空”，“中”，“最亮”，“的”，“星”，“尊敬”，“是”，“我们”，“学习”，“榜样”

attributes	label
1 1 1 1 1 1 1 1 0 0 0 0 0	1
1 1 0 0 0 0 2 0 1 1 1 1 1	1
……	

2.2 TF-IDF

The general idea of TF-IDF is similar to Bag-of-Words, but compared to Bag-of-Words, TF-IDF use inversed document frequency to eliminate the common words like “你” “我” and offers other possible emotional words more opportunity to be involved into the classification. But this is a double-edged sword, as it may also eliminate the common emotional words like “高兴”, therefore we use try both of them in our implementation and choose the one having better accuracy to use.

Now we are going to explain the TF-IDF. Here are the mathematical expressions for calculating the importance and the inversed document frequency:



For a specific word t_i in a specific sentence d_j:

The importance of t_i in d_j $\mathrm{tf_{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}}$ The frequency of word $n_{i,j}$ appears in d_j / The total sum of frequency of all words in d_j

The inversed document frequency of t_i $\mathrm{idf_i} = \lg \frac{|D|}{|\{j : t_i \in d_j\}|}$ The total number of sentence / The total number of sentence having t_i

For a specific word in a specific sentence, we first calculate the importance of word by using its frequency in the sentence divided by the the total sum of frequency of all words in the sentence, this process is on the scale of a single sentence.

Then we obtain the inversed document frequency by use the total number of sentences divided by the total number of sentences having the word, this process is on the scale of all the sentences. It is important to point out that the denominator will never be zero, as all the possible words for the TF-IDF are collected from all the sentence we have, and the minimum of denominator is 1.

At last, we should calculate the TF-IDF value, and here are the mathematical expression:

For a specific word *t_i* in a specific sentence *d_j*:

$$\mathrm{tfidf}_{i,j} = \mathrm{tf}_{i,j} \times \mathrm{idf}_i$$

We get the TF-IDF by the multiplication of the importance and the inversed document frequency, and this step is very important as it connects the result from the scale of single sentence and the scale of all sentences.

3 Implemented Techniques for Sentiment Analysis

There are many available techniques on the sentiment analysis according to the previous survey on Chapter 1.2, but due to time restriction, we cannot implement and introduce all those techniques, as a result, we mainly implemented the Naïve Bayes, SVM and XGBoost. All of them are highly dependent to feature coding, therefore we tried both of the two feature coding techniques Bag-of-Word and TF-IDF. We also used the bagging ensemble learning technique with these three models to increase the accuracy. We have also tried to use Chinese sentiment analysis library SnowNLP as well as the pretrained BERT transformer by Harbin Institute of Technology to do the sentiment analysis, but the result is not used in the latter application due to efficiency problem, therefore SnowNLP and BERT will not be explained here, instead we will explain it in Chapter 4.6 and 4.7.

3.1 Naïve Bayes

The idea of Naïve Bayes mode just like what we learned in class, after using feature encoding techniques, we multiplied the probability of each feature, obtained the prior probability and the posterior probability respectively, compared the value of the two, and finally endowed the value with high probability.

3.2 SVM

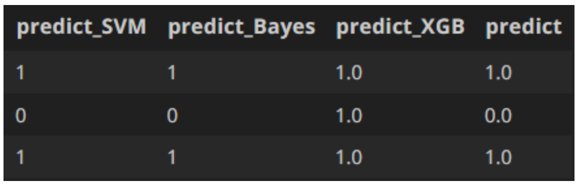
The idea of using SVM in sentiment analysis is also similar to what we have taught, after we done the feature coding, we mark all the sentences on the space, and each of them has a coordinate with a dimension of all the possible words, then we try to find a suitable hyperplane that can divide those sentences into two parts.

3.3 XGBoost

The idea of XGBoost is similar to the decision tree. It construct a decision tree with a subset of all the possible words as splitting attributes, whether having the word or not will result into a subtree or a leaf showing the sentiment. The optimization and construct algorithm for the tree will not be introduced here as they are too difficult.

3.4 Bagging Ensemble Learning

The idea of bagging ensemble learning is simple. Just use Naïve Bayes, SVM, XGBoost to classify the data, and then for each row of data, we choose the prediction appears the most as the final prediction. For example:



predict_SVM	predict_Bayes	predict_XGB	predict
1	1	1.0	1.0
0	0	1.0	0.0
1	1	1.0	1.0

Be attention that the bagging ensemble requires the number of classifier you used to be odd, otherwise there may occur a failure or bias if there are two prediction having the same frequency. That is also another reason why we did not implement any more models.

4 Technical Details for the Implementation

4.1 Raw Dataset

We generally chose three dataset, they are training set, verifying set and predicting set. The training set is for training, and it will be separated to training set and testing set evenly. The verifying set is for verifying the accuracy of the model we trained, and it is collected from a separate source to avoid **overfitting** and **weak generalization ability**. The predicting set is for our application of the models we trained.

As we have mentioned in the introduction, we will do a binary classification of sentiment, and during labeling, we use "1" to represent positive, "0" to represent negative. We also will use this representation in our prediction.

4.1.1 Training Set: 10k labelled weibo in common days of any context.

https://github.com/dengxiuqi/weibo2018

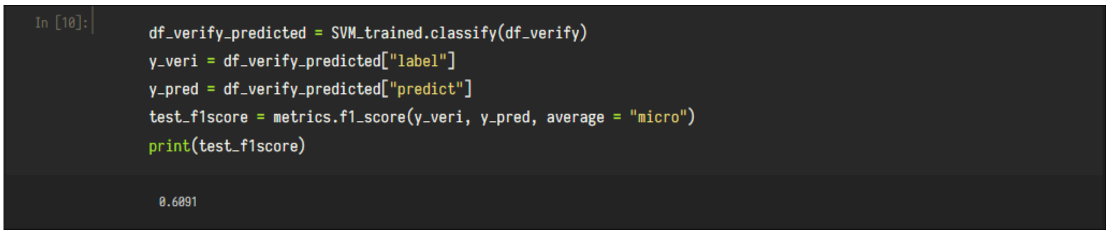
4.1.2 Verifying Set: 100k labelled weibo in common days of any context.

https://github.com/SophonPlus/ChineseNlpCorpus/blob/master/datasets/weibo_senti_100k/intro.ipynb

4.1.3 Predicting Set: 310k weibo collected several month before and during the COVID-19 period (will only be used for the period 2019/09 ~ 2020/04 in our application)

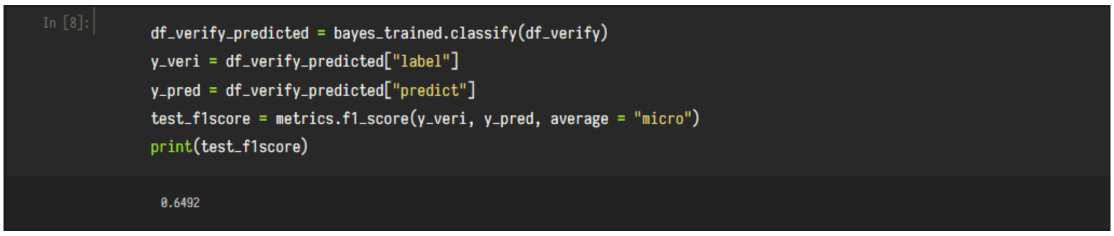
https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DULFFJ

We have tried to use swap the training set and the verifying set, i.e. train our model with 100k weibo data and verify our model with 10k weibo data. But it seems that the 100k weibo data will caused the model to overfit and result in a lack of generalisation ability, the evidence is the low verifying f1score, shown as the graphs below:



```
In [10]:
df_verify_predicted = SVM_trained.classify(df_verify)
y_veri = df_verify_predicted["label"]
y_pred = df_verify_predicted["predict"]
test_f1score = metrics.f1_score(y_veri, y_pred, average = "micro")
print(test_f1score)

0.6091
```

Verify Result for SVM



```
In [8]:
df_verify_predicted = bayes_trained.classify(df_verify)
y_veri = df_verify_predicted["label"]
y_pred = df_verify_predicted["predict"]
test_f1score = metrics.f1_score(y_veri, y_pred, average = "micro")
print(test_f1score)

0.6492
```

Verify Result for Naïve Bayes

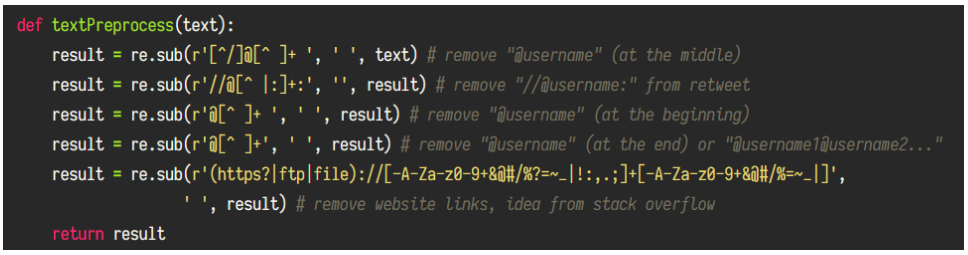
4.2 Data Retrieving for the predicting set

The raw predicting set is 41 json files. Since csv files are more convenient in subsequent data analysis, the dataset is transformed to csv files. And there are some meaningless data whose content is "转发微博" (retweet), these data need to be cleaned. Lastly, the attributes 'content' and 'timestamp' are saved in a file which will using in the further predict.

4.3 Data Preprocessing

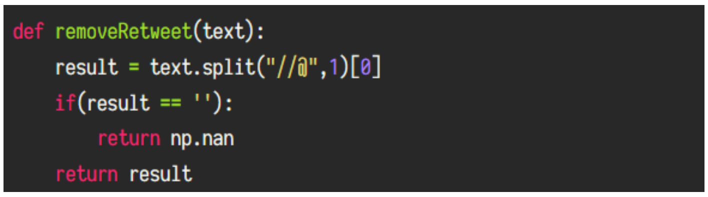
Although we did not collect the data by ourselves, the data we get is still raw data which needs preprocessing to be used on our model.

We first processed the context of the weibo by regular expression in order to remove the unneeded parts.



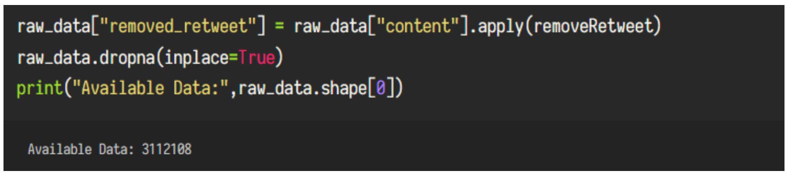
```
def textPreprocess(text):
    result = re.sub(r'[^/]@[^ ]+ ', ' ', text) # remove "@username" (at the middle)
    result = re.sub(r'//@[^ |:]+:', '', result) # remove "//@username:" from retweet
    result = re.sub(r'@[^ ]+ ', ' ', result) # remove "@username" (at the beginning)
    result = re.sub(r'@[^ ]+', ' ', result) # remove "@username" (at the end) or "@username1@username2..."
    result = re.sub(r'(https?|ftp|file)://[-A-Za-z0-9+&@#/%?=~_|!:,.;]+[-A-Za-z0-9+&@#/%=~_|]',
                    ' ', result) # remove website links, idea from stack overflow
    return result
```

We keep the retweets for the testing set and the verifying set, as their labels all includes the sentiment of retweet, but be remove the retweets for the predicting set as we want to obtain an accurate result.



```
def removeRetweet(text):
    result = text.split("//@",1)[0]
    if(result == ''):
        return np.nan
    return result
```

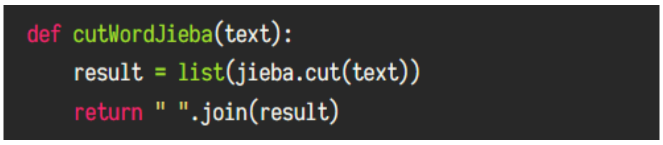
After removing the retweet, there may exist NaN context weibo, therefore we also need to popped out those NaN weibo.



```
raw_data["removed_retweet"] = raw_data["content"].apply(removeRetweet)
raw_data.dropna(inplace=True)
print("Available Data:",raw_data.shape[0])
```

```
Available Data: 3112108
```

At last, since Naïve Bayes, SVM and XGBoost all need the context to be cut, we used the Jieba to do a word cut in advance and output a word cut version of the dataset.



```
def cutWordJieba(text):
    result = list(jieba.cut(text))
    return " ".join(result)
```

4.4 Data used for model

The processed dataset will be uploaded to github since it is too big to be uploaded among the iSpace, the link is: https://github.com/lutianyu2001/data_mining_project

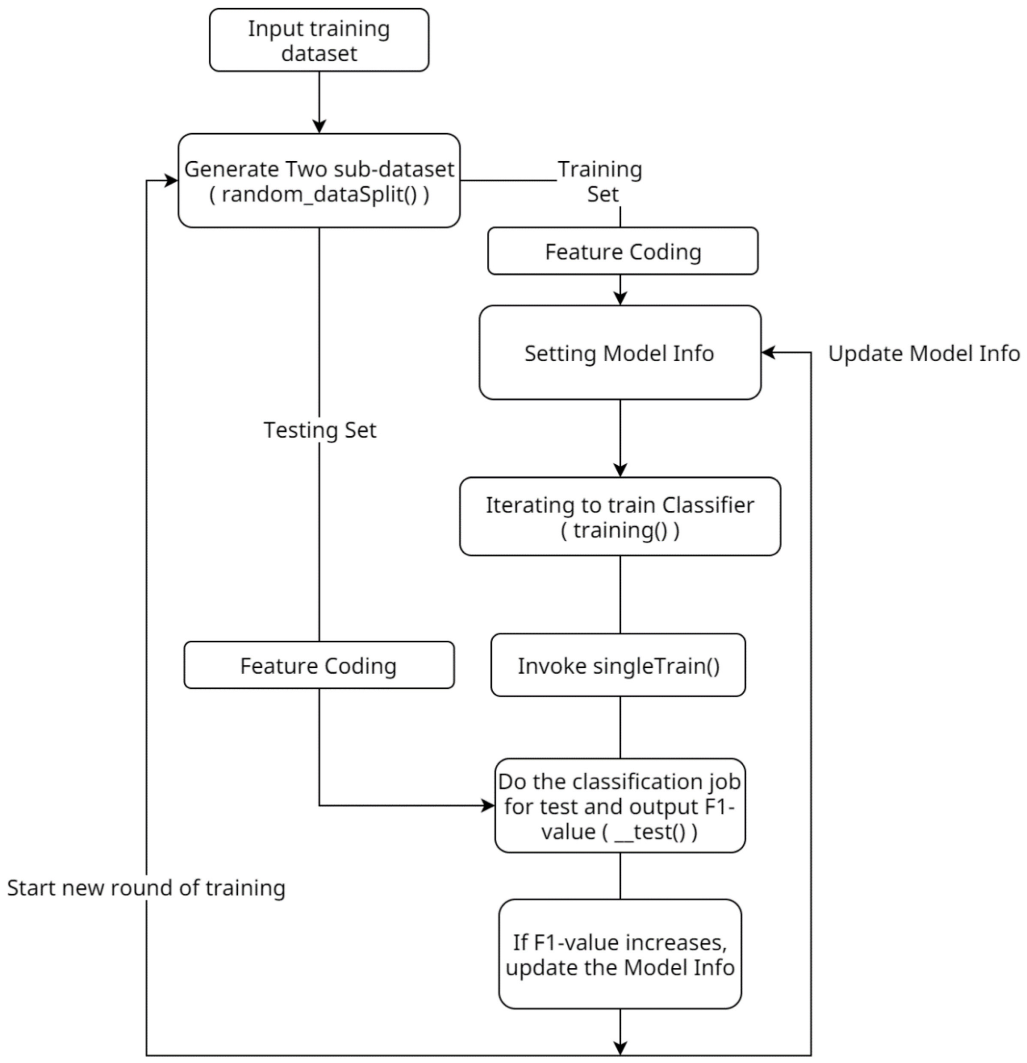
4.5 Implementations for Naïve Bayes, SVM and XGBoost

We will not explain our code here, as our comments in the Jupyter notebook has already explained in detail. We have written a basic framework for those models to have an integrated training system, which has these functions.

sentiment_[model]

- classify()
- training()
- singleTrain()
- random_dataSplit()
- TFIDFcoding()
- BOWcoding()
- __test()

And it will operate as the workflow for training:


Input training dataset
Generate Two sub-dataset (random_dataSplit())
Training Set
Feature Coding
Setting Model Info
Update Model Info
Testing Set
Iterating to train Classifier (training())
Feature Coding
Invoke singleTrain()
Do the classification job for test and output F1-value (__test())
Start new round of training
If F1-value increases, update the Model Info

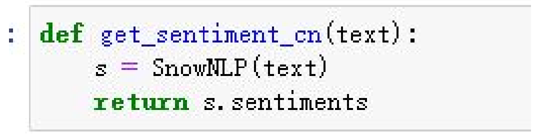
The method for feature coding can be specified by user. Meanwhile, the classify() method is for classifying a given predicting set.

We trained all the models we have, and the f1score for all of them are near 0.82, and for the feature coding, both Naïve Bayes and XGBoost use BoW, and SVM uses TF-IDF. At last we used the bagging ensemble learning (in Data Mining Proj_Bagging.ipynb), which has a f1score of 0.84 on the verify set, clearly bagging increases the accuracy of the model.

4.6 Implementations for SnowNLP

SnowNLP is a class library written in python to facilitate the processing of Chinese text content, which was inspired by TextBlob. All of the algorithms are implemented by the library itself, and it comes with some trained dictionaries.

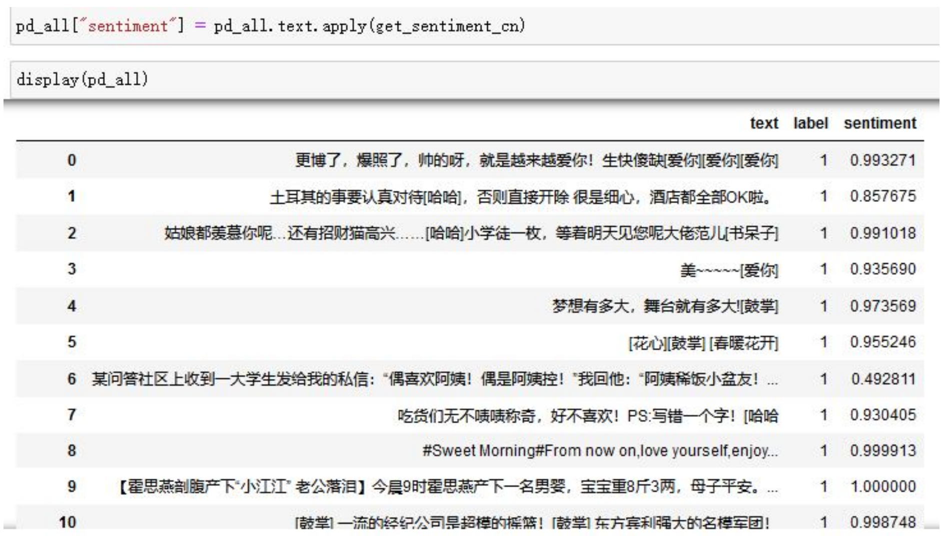
Then we can use the function to do the sentiment analysis



```
def get_sentiment_cn(text):
    s = SnowNLP(text)
    return s.sentiments
```

We set if the value is closer to 1 means positive emotion, closer to 0 means negative emotion.

Then we can get the result for the training data:



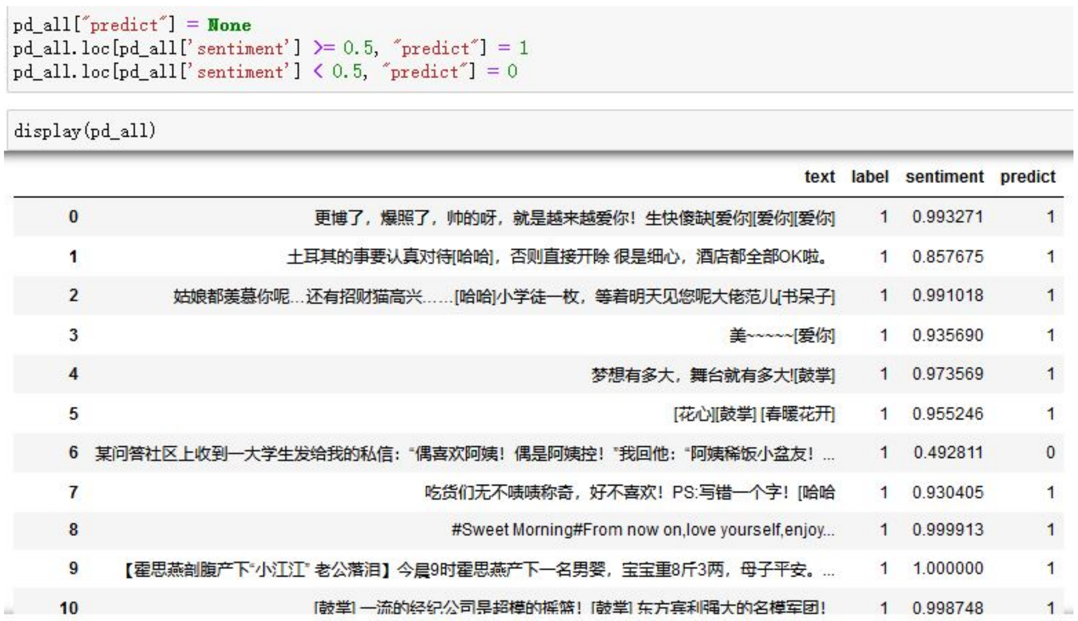
```
pd_all["sentiment"] = pd_all.text.apply(get_sentiment_cn)
```

```
display(pd_all)
```

	text	label	sentiment
0	更博了，爆照了，帅的呀，就是越来越爱你！生快傻缺[爱你][爱你][爱你]	1	0.993271
1	土耳其的事要认真对待[哈哈]，否则直接开除 很是细心，酒店都全部OK啦。	1	0.857675
2	姑娘都羡慕你呢…还有招财猫高兴……[哈哈]小学徒一枚，等着明天见您呢大佬范儿[书呆子]	1	0.991018
3	美~~~~~[爱你]	1	0.935690
4	梦想有多大，舞台就有多大![鼓掌]	1	0.973569
5	[花心][鼓掌] [春暖花开]	1	0.955246
6	某问答社区上收到一大学生发给我的私信："偶喜欢阿姨！偶是阿姨控！"我回他："阿姨稀饭小盆友！ ...	1	0.492811
7	吃货们无不啧啧称奇，好不喜欢！PS:写错一个字！[哈哈	1	0.930405
8	#Sweet Morning#From now on,love yourself,enjoy...	1	0.999913
9	【霍思燕剖腹产下"小江江" 老公落泪】今晨9时霍思燕产下一名男婴，宝宝重8斤3两，母子平安。...	1	1.000000
10	[鼓掌] 一流的经纪公司是超模的摇篮！[鼓掌] 东方宾利强大的名模军团！	1	0.998748

In the result, we can find that for different sentences, we can get different results

Then we need to update the label value and create a new columns to record the new value which name is name is predict value, if the sentiment value is larger or equal to 0.5, the predict value is 1; if the value is smaller than 0.5, the predict value is 0. Here we can get our new result.

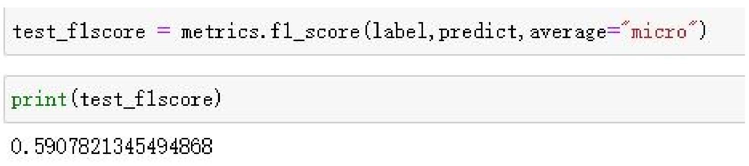


```
pd_all["predict"] = None
pd_all.loc[pd_all['sentiment'] >= 0.5, "predict"] = 1
pd_all.loc[pd_all['sentiment'] < 0.5, "predict"] = 0
```

```
display(pd_all)
```

	text	label	sentiment	predict
0	更博了，爆照了，帅的呀，就是越来越爱你！生快傻缺[爱你][爱你][爱你]	1	0.993271	1
1	土耳其的事要认真对待[哈哈]，否则直接开除 很是细心，酒店都全部OK啦。	1	0.857675	1
2	姑娘都羡慕你呢…还有招财猫高兴……[哈哈]小学徒一枚，等着明天见您呢大佬范儿[书呆子]	1	0.991018	1
3	美~~~~~[爱你]	1	0.935690	1
4	梦想有多大，舞台就有多大![鼓掌]	1	0.973569	1
5	[花心][鼓掌] [春暖花开]	1	0.955246	1
6	某问答社区上收到一大学生发给我的私信："偶喜欢阿姨！偶是阿姨控！"我回他："阿姨稀饭小盆友！ ...	1	0.492811	0
7	吃货们无不啧啧称奇，好不喜欢！PS:写错一个字！[哈哈	1	0.930405	1
8	#Sweet Morning#From now on,love yourself,enjoy...	1	0.999913	1
9	【霍思燕剖腹产下"小江江" 老公落泪】今晨9时霍思燕产下一名男婴，宝宝重8斤3两，母子平安。...	1	1.000000	1
10	[鼓掌] 一流的经纪公司是超模的摇篮！[鼓掌] 东方宾利强大的名模军团！	1	0.998748	1

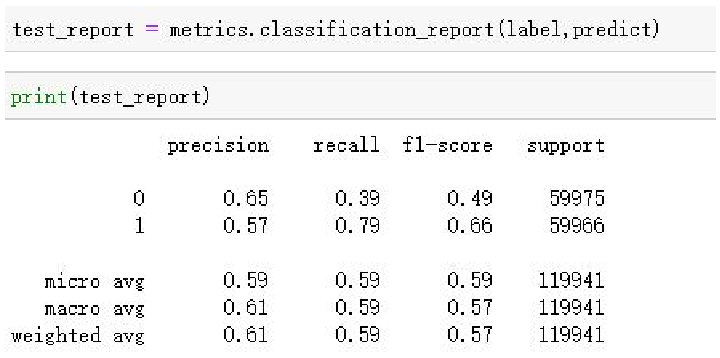
Next we need to calculate its F1-score and get the report



```
test_f1score = metrics.f1_score(label,predict,average="micro")
```

```
print(test_f1score)
```

```
0.5907821345494868
```



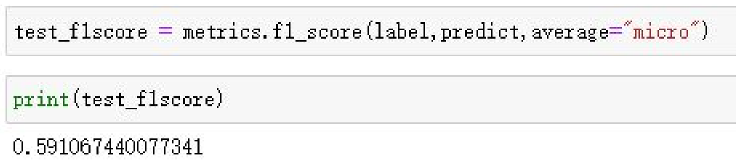
```
test_report = metrics.classification_report(label,predict)
```

```
print(test_report)
```

```
              precision    recall  f1-score   support

           0       0.65      0.39      0.49     59975
           1       0.57      0.79      0.66     59966

   micro avg       0.59      0.59      0.59    119941
   macro avg       0.61      0.59      0.57    119941
weighted avg       0.61      0.59      0.57    119941
```

In this we can find the F1-score is 0.59, it seems that it is very low, the reason for this problem is that the models already trained in the library are based on product review data, so using the trained models to analysis the tweet data will result in errors, leading to a low accuracy rate in the end.

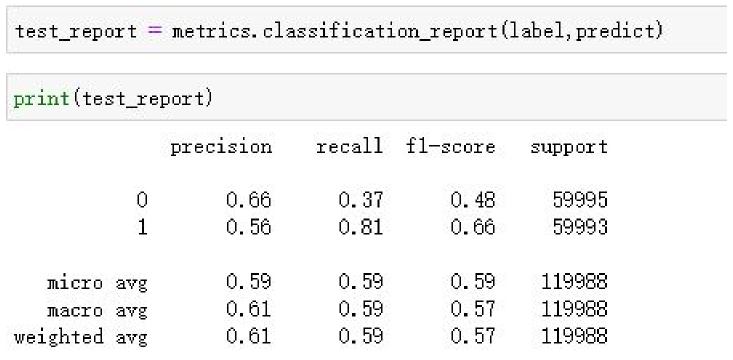
Then we do the same analysis for the WordCut data, here is the F1-score and the report



```
test_f1score = metrics.f1_score(label,predict,average="micro")
```

```
print(test_f1score)
```

```
0.591067440077341
```



```
test_report = metrics.classification_report(label,predict)
```

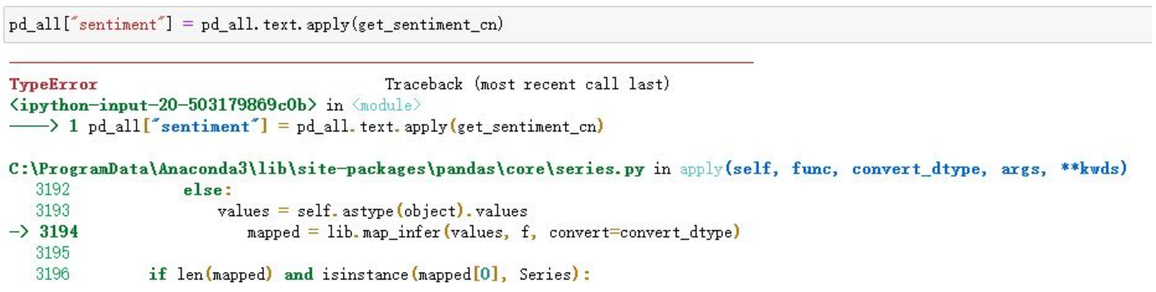
```
print(test_report)
```

```
              precision    recall  f1-score   support

           0       0.66      0.37      0.48     59995
           1       0.56      0.81      0.66     59993

   micro avg       0.59      0.59      0.59    119988
   macro avg       0.61      0.59      0.57    119988
weighted avg       0.61      0.59      0.57    119988
```

We find that the accuracy is low, too.

Thus, the SnowNLP model is not a good model for our data.

Next we use the model to analysis the test data, after seven hours, we get the error:


```
pd_all["sentiment"] = pd_all.text.apply(get_sentiment_cn)
```

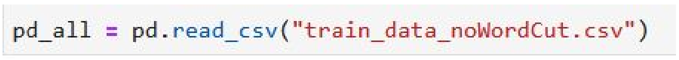
```
TypeError                                 Traceback (most recent call last)
<ipython-input-20-503179869c0b> in <module>
----> 1 pd_all["sentiment"] = pd_all.text.apply(get_sentiment_cn)

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\series.py in apply(self, func, convert_dtype, args, **kwds)
   3192             else:
   3193                 values = self.astype(object).values
-> 3194                 mapped = lib.map_infer(values, f, convert=convert_dtype)
   3195 
   3196         if len(mapped) and isinstance(mapped[0], Series):
```

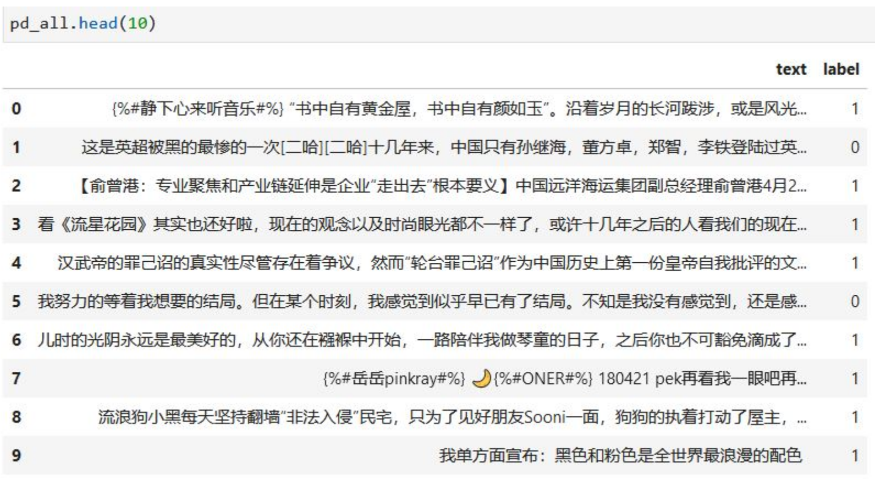
Therefore we not use this model in our application.

4.7 Implementations for BERT

Bert, known as Bidirectional Encoder Representation from Transformers, is a pre-trained language representation model. It emphasizes the use of the new masked language model (MLM) instead of the traditional unidirectional language model or the shallow splicing of two unidirectional language models for pretraining, as in the past, so that deep bidirectional language representations can be generated. At the time of publication, the BERT paper mentioned that new state-of-the-art results were obtained in 11 NLP (Natural Language Processing) tasks. The model has the following main advantages: MLM is used to pre-train bi-directional Transformers to generate deep bi-directional language representations. After pre-training, only an additional output layer needs to be added for fine-tune to achieve state-of-the-art performance in a wide variety of downstream tasks. Let's proceed with the details according to code.

The training data is first read as our test set.

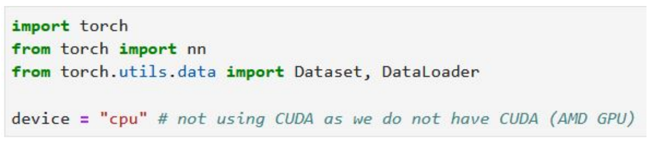

```
pd_all = pd.read_csv("train_data_noWordCut.csv")
```



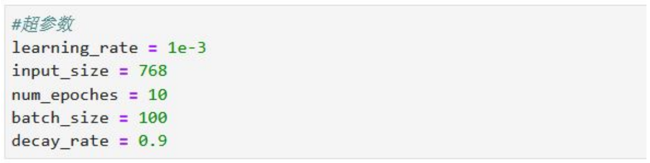
```
pd_all.head(10)
```

	text	label
0	{%#静下心来听音乐#%} "书中自有黄金屋，书中自有颜如玉"。沿着岁月的长河跋涉，或是风光...	1
1	这是英超被黑的最惨的一次[二哈][二哈]十几年来，中国只有孙继海，董方卓，郑智，李铁登陆过英...	0
2	【俞曾港：专业聚焦和产业链延伸是企业"走出去"根本要义】中国远洋海运集团副总经理俞曾港4月2...	1
3	看《流星花园》其实也还好啦，现在的观念以及时尚眼光都不一样了，或许十几年之后的人看我们的现在...	1
4	汉武帝的罪己诏的真实性尽管存在着争议，然而"轮台罪己诏"作为中国历史上第一份皇帝自我批评的文...	1
5	我努力的等着我想要的结局。但在某个时刻，我感觉到似乎早已有了结局。不知是我没有感觉到，还是感...	0
6	儿时的光阴永远是最美好的，从你还在襁褓中开始，一路陪伴我做琴童的日子，之后你也不可豁免滴成了...	1
7	{%#岳岳pinkray#%} 🌙{%#ONER#%} 180421 pek再看我一眼吧再...	1
8	流浪狗小黑每天坚持翻墙"非法入侵"民宅，只为了见好朋友Sooni一面，狗狗的执着打动了屋主，...	1
9	我单方面宣布：黑色和粉色是全世界最浪漫的配色	1

The next step is to set the parameters, because we don't have CUDA, so we won't use it. Instead, some other parameters are set to facilitate the use of the function for a bit.

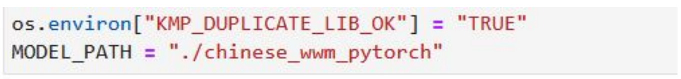


```
import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader

device = "cpu" # not using CUDA as we do not have CUDA (AMD GPU)
```

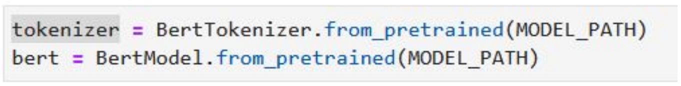


```
#超参数
learning_rate = 1e-3
input_size = 768
num_epoches = 10
batch_size = 100
decay_rate = 0.9
```

For the model path, we use "chinese_wwm_pytorch", which is a model for Chinese provided on GitHub for this model, so our separator and model are fetched from this path for the next calculation.

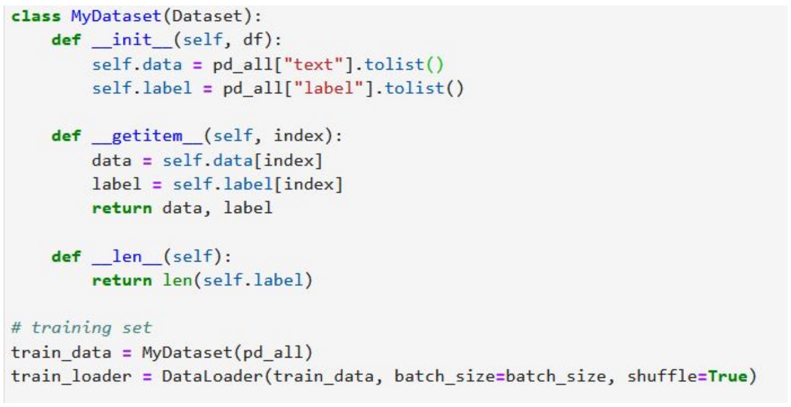


```
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"
MODEL_PATH = "./chinese_wwm_pytorch"
```



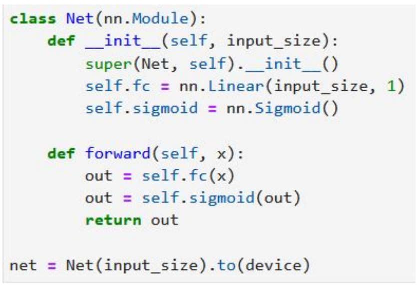
```
tokenizer = BertTokenizer.from_pretrained(MODEL_PATH)
bert = BertModel.from_pretrained(MODEL_PATH)
```

Read the data and put our data into the model for training.



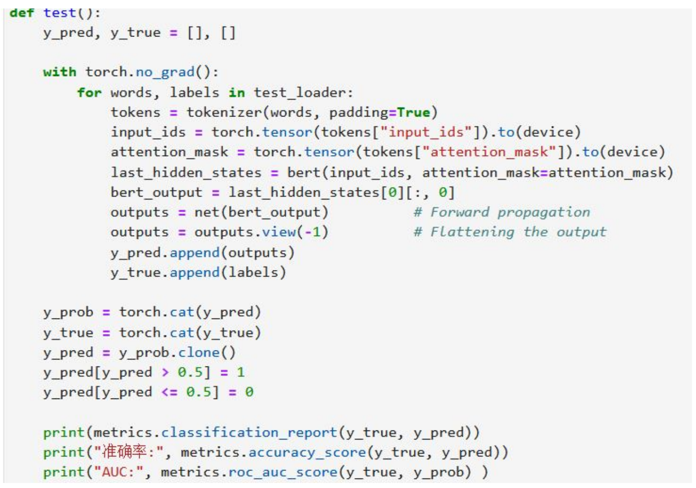
```
class MyDataset(Dataset):
    def __init__(self, df):
        self.data = pd_all["text"].tolist()
        self.label = pd_all["label"].tolist()

    def __getitem__(self, index):
        data = self.data[index]
        label = self.label[index]
        return data, label

    def __len__(self):
        return len(self.label)

# training set
train_data = MyDataset(pd_all)
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
```

Building neural networks with the pytorch framework. For the __ init__ () function, after creating a class in python, a __ init__ () method is usually created, which is automatically executed when an instance of the class is created. The __ init__ () method must contain a self parameter, and it must be the first parameter. Super(Net, self)._init_() in python is to first find the parent class of Net, and then convert the object self of class Net to the object of class NNet, and then the "converted" class Net object calls its own _init_ function. In fact, the simple understanding is that the subclasses put the __init__() of the parent class into their own __init__(), so that the class has the __init__() of the parent class.



```
class Net(nn.Module):
    def __init__(self, input_size):
        super(Net, self).__init__()
        self.fc = nn.Linear(input_size, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        out = self.fc(x)
        out = self.sigmoid(out)
        return out

net = Net(input_size).to(device)
```

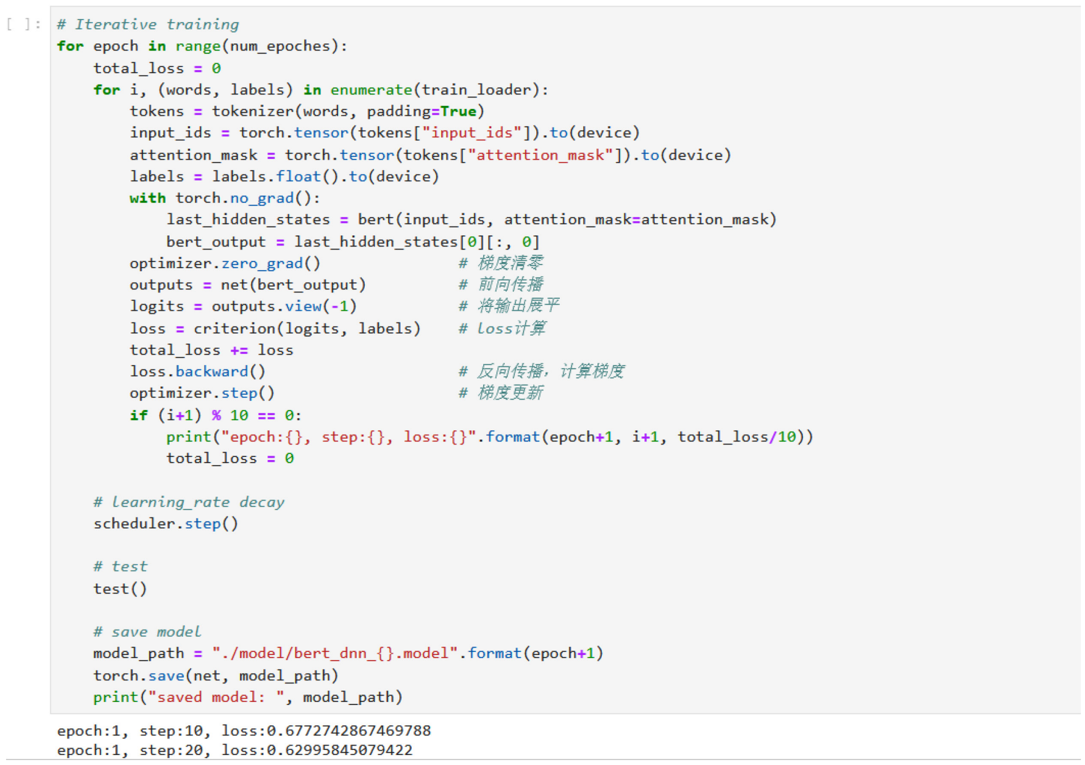
Test set effect test, to facilitate subsequent iterations of the training output results, to judge.



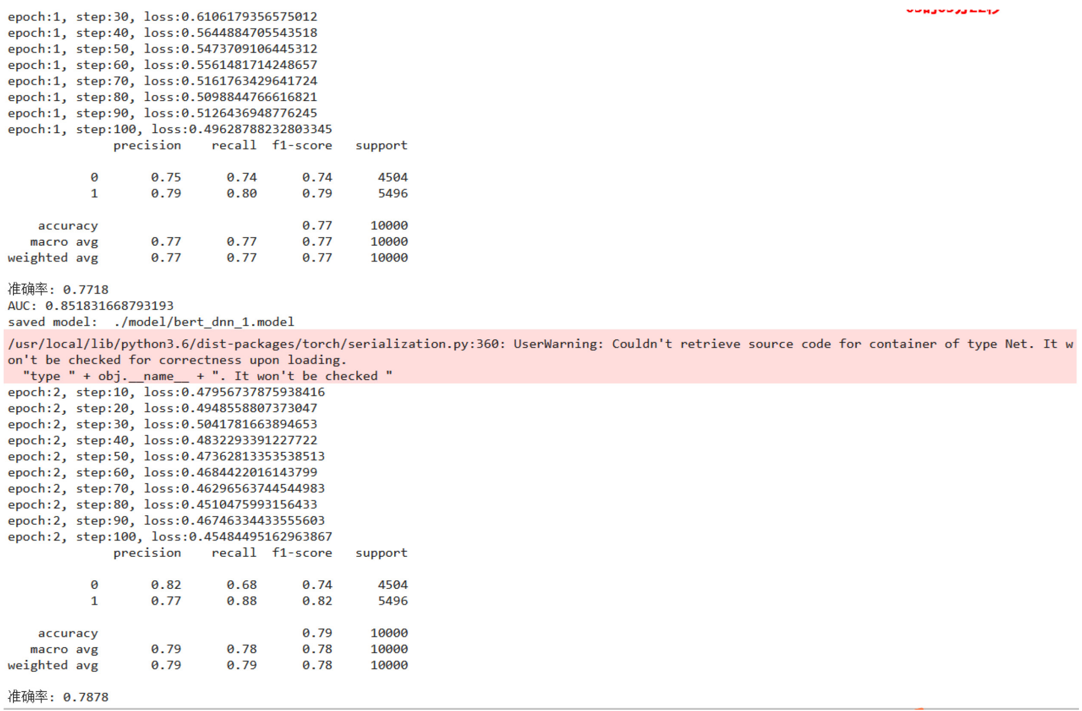
```
def test():
    y_pred, y_true = [], []

    with torch.no_grad():
        for words, labels in test_loader:
            tokens = tokenizer(words, padding=True)
            input_ids = torch.tensor(tokens["input_ids"]).to(device)
            attention_mask = torch.tensor(tokens["attention_mask"]).to(device)
            last_hidden_states = bert(input_ids, attention_mask=attention_mask)
            bert_output = last_hidden_states[0][:, 0]
            outputs = net(bert_output)          # Forward propagation
            outputs = outputs.view(-1)          # Flattening the output
            y_pred.append(outputs)
            y_true.append(labels)

    y_prob = torch.cat(y_pred)
    y_true = torch.cat(y_true)
    y_pred = y_prob.clone()
    y_pred[y_pred > 0.5] = 1
    y_pred[y_pred <= 0.5] = 0

    print(metrics.classification_report(y_true, y_pred))
    print("准确率:", metrics.accuracy_score(y_true, y_pred))
    print("AUC:", metrics.roc_auc_score(y_true, y_prob) )
```

The best model is used to calculate the predict data, and we have trained three models.

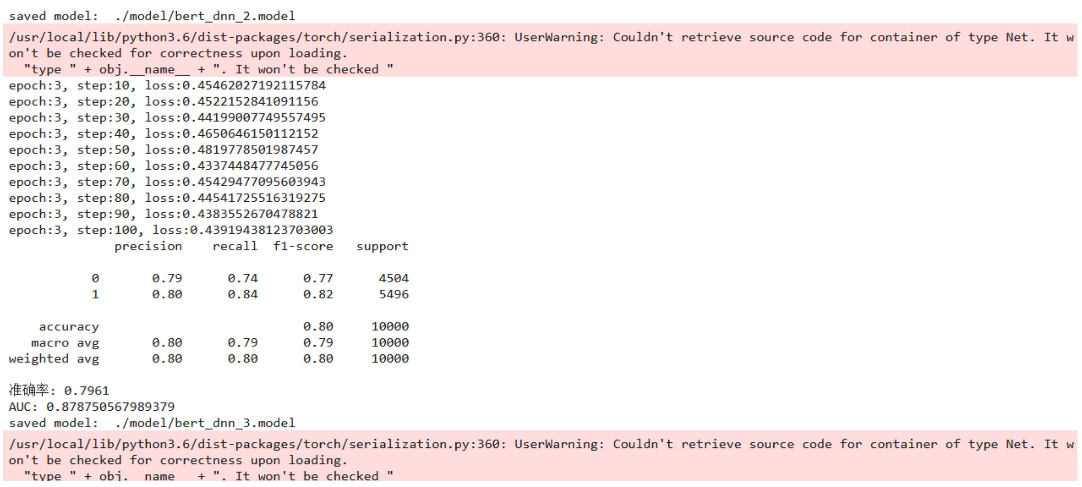
Here is our final result and the three models:



```
[ ]: # Iterative training
     for epoch in range(num_epoches):
         total_loss = 0
         for i, (words, labels) in enumerate(train_loader):
             tokens = tokenizer(words, padding=True)
             input_ids = torch.tensor(tokens["input_ids"]).to(device)
             attention_mask = torch.tensor(tokens["attention_mask"]).to(device)
             labels = labels.float().to(device)
             with torch.no_grad():
                 last_hidden_states = bert(input_ids, attention_mask=attention_mask)
                 bert_output = last_hidden_states[0][:, 0]
             optimizer.zero_grad()              # 梯度清零
             outputs = net(bert_output)         # 前向传播
             logits = outputs.view(-1)          # 将输出展平
             loss = criterion(logits, labels)   # loss计算
             total_loss += loss
             loss.backward()                    # 反向传播，计算梯度
             optimizer.step()                   # 梯度更新
             if (i+1) % 10 == 0:
                 print("epoch:{}, step:{}, loss:{}".format(epoch+1, i+1, total_loss/10))
                 total_loss = 0

         # learning_rate decay
         scheduler.step()

         # test
         test()

         # save model
         model_path = "./model/bert_dnn_{}.model".format(epoch+1)
         torch.save(net, model_path)
         print("saved model: ", model_path)

     epoch:1, step:10, loss:0.6772742867469788
     epoch:1, step:20, loss:0.62995845079422
```



```
epoch:1, step:30, loss:0.6106179356575012
epoch:1, step:40, loss:0.5644884705543518
epoch:1, step:50, loss:0.5473709106445312
epoch:1, step:60, loss:0.5561481714248657
epoch:1, step:70, loss:0.5161763429641724
epoch:1, step:80, loss:0.5098844766616821
epoch:1, step:90, loss:0.5126436948776245
epoch:1, step:100, loss:0.49628788232803345
              precision    recall  f1-score   support

           0       0.75      0.74      0.74      4504
           1       0.79      0.80      0.79      5496

    accuracy                           0.77     10000
   macro avg       0.77      0.77      0.77     10000
weighted avg       0.77      0.77      0.77     10000

准确率: 0.7718
AUC: 0.851831668793193
saved model:  ./model/bert_dnn_1.model
/usr/local/lib/python3.6/dist-packages/torch/serialization.py:360: UserWarning: Couldn't retrieve source code for container of type Net. It w
on't be checked for correctness upon loading.
  "type " + obj.__name__ + ". It won't be checked "
epoch:2, step:10, loss:0.47956737875938416
epoch:2, step:20, loss:0.4948558807373047
epoch:2, step:30, loss:0.5041781663894653
epoch:2, step:40, loss:0.4832293391227722
epoch:2, step:50, loss:0.47362813353538513
epoch:2, step:60, loss:0.4684422016143799
epoch:2, step:70, loss:0.46296563744544983
epoch:2, step:80, loss:0.4510475993156433
epoch:2, step:90, loss:0.46746334433555603
epoch:2, step:100, loss:0.45484495162963867
              precision    recall  f1-score   support

           0       0.82      0.68      0.74      4504
           1       0.77      0.88      0.82      5496

    accuracy                           0.79     10000
   macro avg       0.79      0.78      0.78     10000
weighted avg       0.79      0.79      0.78     10000

准确率: 0.7878
```



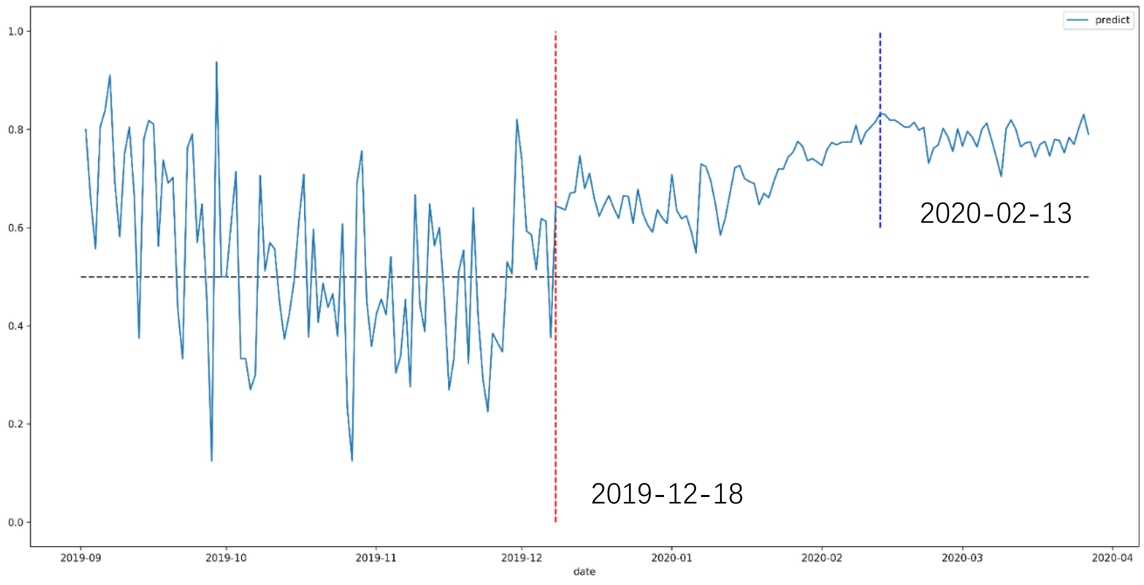
```
saved model:  ./model/bert_dnn_2.model
/usr/local/lib/python3.6/dist-packages/torch/serialization.py:360: UserWarning: Couldn't retrieve source code for container of type Net. It w
on't be checked for correctness upon loading.
  "type " + obj.__name__ + ". It won't be checked "
epoch:3, step:10, loss:0.45462027192115784
epoch:3, step:20, loss:0.4522152841091156
epoch:3, step:30, loss:0.44199007749557495
epoch:3, step:40, loss:0.4650646150112152
epoch:3, step:50, loss:0.4819778501987457
epoch:3, step:60, loss:0.4337448477745056
epoch:3, step:70, loss:0.45429477095603943
epoch:3, step:80, loss:0.44541725516319275
epoch:3, step:90, loss:0.4383552670478821
epoch:3, step:100, loss:0.43919438123703003
              precision    recall  f1-score   support

           0       0.79      0.74      0.77      4504
           1       0.80      0.84      0.82      5496

    accuracy                           0.80     10000
   macro avg       0.80      0.79      0.79     10000
weighted avg       0.80      0.80      0.80     10000

准确率: 0.7961
AUC: 0.878750567989379
saved model:  ./model/bert_dnn_3.model
/usr/local/lib/python3.6/dist-packages/torch/serialization.py:360: UserWarning: Couldn't retrieve source code for container of type Net. It w
on't be checked for correctness upon loading.
  "type " + obj.__name__ + ". It won't be checked "
```

In the result, we can find that model 3 has the highest accuracy rate. So, we use model 3 as our base model for the validation set.

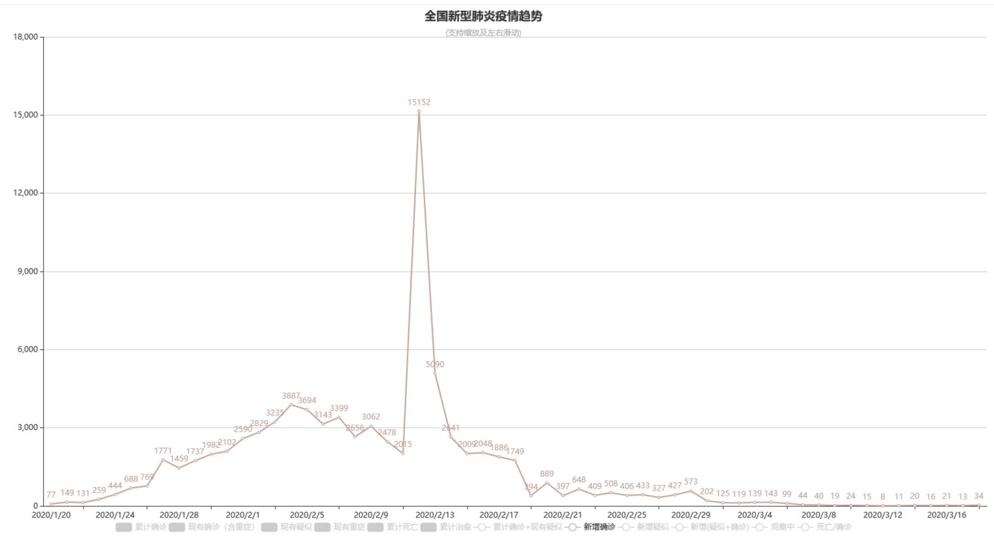
But in the computation, our kernel will die unexpectedly and cannot proceed to the next step to get the result, so we abandon the BERT model.

5 Application & Result Analysis

We apply our models to the predicting set, calculate everyday average value of the prediction, and use matplotlib() to do the time series visualisation, then we get this plot, which shows the changes in average sentiment from 2019-09-01 to 2020-04-01:



The first COVID-19 appears at 2019-12-18, we can see that people's sentiment fluctuates acutely as people's emotion are not connected. But soon after the COVID-19 appears, people's emotion are connected and are positive because they believe in our government. Meanwhile, there occurs a small increase in the newly confirmed cases of COVID-19 at 2020-02-13 (shown as the graph below), therefore the sentiment has a small fluctuation after that day.


全国新型肺炎疫情趋势
(支持缩放及左右滑动)
2020/1/20
2020/1/24
2020/1/28
2020/2/1
2020/2/5
2020/2/9
2020/2/13
2020/2/17
2020/2/21
2020/2/25
2020/2/29
2020/3/4
2020/3/8
2020/3/12
2020/3/16
累计确诊
现有确诊（含重症）
现有疑似
现有重症
累计死亡
累计治愈
累计确诊+现有疑似
新增确诊
新增疑似
新增(疑似+确诊)
观察中
死亡/确诊