QBUS6850 Machine Learning for Business

(2022S1)

Sentiment Analysis on Amazon Kindle Book Review using Bi-LSTM-TextCNN Structure and Ordinal Regression

Group Number: 12

Group Members:

510170981

510207449

480397865

510160513

500322217

Abstract

This research is conducting sentiment analysis based on Amazon Kindel Book Review, which is an emerging field with prospecting business value. After understanding the data and reasonable cleaning process, several possibilities are explored. Combining TF-IDF with the gradient boosting classification model has shown to have the best performance. A novel neural network structure of combing Bi-LSTM and TextCNN is experimented with but does not provide a satisfying result. The ordinal regression method is also experimented with in the network, expected to avoid information loss compared with traditional classification and regression. A state of the art technique of using BERT also failed to outperform the traditional one. In the end, we concluded the recommendation of the models and reflected on the experiment details.

1. Introduction

The emerging digital businesses and online social network platforms have been creating a vast amount of text data, including social media posts, product reviews, answers to surveys, etc. In the context of the information era, these platforms seem to be the major channel for people expressing their thoughts and opinions toward certain topics and commodities, which contains valuable information that can facilitate the businesses to understand the market reaction, predict the trend and shed light on product innovation. (Saad & Yang, 2019) (Liu & Xu, 2021)

However, extracting information from these unstructured data can be challenging because the amount of text is beyond human-level manipulation and complexity is embedded in the various form of human language. This is where advanced natural language processing techniques and tools - especially sentiment analysis, also known as opinion mining - are specialised in. Sentiment analysis refers to the process of investigating the explicit and implicit opinions of the users based on the textual information generated. (Fersini et al., 2017) As an emerging field, sentiment analysis has been noticed and valued increasingly by both research and industry practitioners. (Mäntylä, Graziotin & Kuutila, 2018)

Sentiment analysis represents a series of methods, tools and techniques. And these methods have changed from statistical models(such as the Naive Bayes method) to advanced machine learning tasks (such as deep learning). The goals of sentiment analysis have also evolved from polarized opinion(positive, neutral, negative) to a more fine-grained manner(categorizing with the strength of sentiment and also other tasks like topic extraction).(Mäntylä, Graziotin & Kuutila, 2018)

Sentiment analysis is highly dependent on the context. Some practices conducting sentiment analysis toward Twitter posts have already shown a satisfying accuracy which is more focused on social media text data. (Nasukawa & Yi, 2003) In this study, the Amazon kindle book review dataset is used to build sentiment analysis models in order to match the textual comments on books with its rating. Such analysis can be served as a guide for the publishing industry to deeply understand the opinions of readers instead of merely numerical ratings.

The rest of the study is about: Section 2 is about related work in natural language processing and advanced model used. Section 3 is the details of the model involved in this study. Section 4 specifies the details of the experiments and also the results. Finally, section 5 summarizes the study and reflects on the experiments.

2. Literature review

Sentiment analysis requires techniques in various aspects of natural language processing and machine learning. Relevant research has greatly advanced since 2004.

As a typical workflow, the starting point of sentiment analysis by the analysts begins with handling the collected text data to transform them into a cleaned one. Researchers looked into different text preprocessing techniques. Noises in text data include HTML tags, URLs, informal languages, misused punctuations etc., which have little information but add the difficulty for the features to be extracted from the text. Moreover, A careful handling of emoticons, semantic negation, dictionary of misspelling, lemmatization and stopwords have been proven to enhance the performance of language models.(Giulio et al., 2012) POS(Part of speech) tagging focus on categorizing the words based on grammar, which can furthermore improve the accuracy of sentence parsing.(Slav et al., 2011)

Another big challenge for sentiment analysis is feature extraction and selection. Compared with more common number based models, sentiment analysis deals with a large number of words that are hard to be precisely summarised as numeric presentation. It is mentioned that there are different directions for feature extraction.(Muhammad Z.A. et al., 2014) Morphological methods including semantic, syntactic and lexicon structures consider the context and try to extract information as humans understand the text. Frequency based features are using statistical methods and calculations to build features based on the frequency of a word in the text. Relevant techniques include bag of words(BOW), and Term Frequency Inverse Document Frequency(TF-IDF). Problems in these feature extractions are the high dimension of variables and the subsequent sparse data.

Feature extraction and text representation benefit from the rising neural network and deep learning research. Embedding is able to create continuous and low-dimensional vectors for each word, advances in the field include creating embeddings for sentences and even documents. (Moghadasi & Zhuang, 2020) Attention based transformers are a great leap in turning textual data into numbers, which outperforms many old techniques.(Ashish V. et al, 2017)

The last step of choosing the right model has also greatly developed with neural networks. Naive Bayes and Support Vector Machine performed quite well in sentiment classification. (Srujan et al., 2018) Recent studies demonstrate to lay the foundation to perform hierarchical classification combining convolutional neural network (CNN) and recurrent neural network (RNN). (Arief et al., 2022). Sangeetha and Prabha (2021) produced novel attention models based on LSTM layers to longer sequences of sentences and address the referral word. Chen et al. (2020) also produced similar findings where the application of the deep learning model LSTM improved both prediction accuracy and F1 measure.

3. Description and discussion of the models being investigated

3.1. TASK A: TF-IDF - Gradient boosting

3.1.1 Feature Engineering - TF-IDF

TF-IDF is used to numerically represent the text, which can reflect the importance of the word in the document by calculating the product of term frequency and inverse document frequency. The larger TF-IDF is, the more important the word is in the document. This method is used because it has the advantage of being convenient and quick to convert text data and easy to understand.

Compared with BoW, which simply counts the unique words and stores the frequencies, TF-IDF considers not only the frequency of the word but also the document. Such that rare but important words would have a higher value and frequent but useless words(such as ‘I’, ‘to’, ‘the’) would have a much lower value.

3.1.2 Model - Gradient Boosting

Gradient boosting is used for modelling. It is an algorithm that continuously reduces the residual value during training. This method can assemble multiple week learners, assign different weights according to their performance, and continuously improve the entire model. Both classification and regression have similar processes. The first step is to calculate the negative gradient, the second step is to fit a simple model(decision stump or slightly more complicated ones) to the negative gradient, then ensemble the weak one to become the updated model. GridsearchCV is used for selecting the optimised hyperparameters.

In the result, regression gets numerical variables, and classification gets a categorical prediction. Considering this is a multi-classification task with numerical output, the result of the regression needs to be converted back into integer prediction, which will be elaborated on in a later section.

The regression model performance is evaluated by mean squared error, and its converted categorial prediction is evaluated by the F1- score and accuracy score. The classification model performance is calculated by macro F1-score and accuracy score.

3.1.3 Setting Thresholds for the Output in Regression

One alternative way of converting the continuous output is to round the result to the nearest 1, 2, 3, 4, 5 level, for example, any result less than 1.5 is classified as 1, and any greater than 4.5 is classified as 5. The results of this division show that there are very few samples with a grade of 5. In the experiment, only 93 of the 2250 test data were divided into a grade of 5.

This obviously contradicts our EDA where in rating 5 there are more samples. Although the assumption in distribution is not strict, each level of the result should not be too small. Therefore, the threshold here adopts the method of conforming to the previous level's distribution. In other words, the proportion of each level in the training set is calculated and mapped these 'percentiles' into predicted results, to divide them into 5 classes. This technique will be introduced more specifically in experiment details.

For example, if rating 1 accounts for 18%, then sort the predicted results from small to large, and any number less than 18 percentile is classified in rating 1. Similarly, if rating 2 accounts for 35%, calculate a percentile of 35% plus 18% which is 53% and any number less than 53 percentile in the output space and greater than 18 percentile will be divided into rating 2, and so on. In this way, all numbers are converted into ratings and have a reasonable distribution.

3.2. TASK B - Glove - BiLSTM - TextCNN - Ordinal Regression

3.2.1 Embedding Layer - Glove

Considering the TF-IDF method of feature engineering in Task A creates too many dimensions and discards important information about the meaning of words in the context as it breaks the order of the words in the document.

Global Vectors for Word Representation (GloVe) embedding is used to be the vectorized representation for the words in each document. The advantage of using GloVe is that it is able to capture the relationship between different words. Moreover, GloVe trains the weights based on the entire corpus which deals with the problem of some words' rare occurrences. With pre-trained weights, GloVe reduces the bias caused by small datasets as there are only 9000 examples provided.

3.2.2 Bi-LSTM

Bidirectional long short-term memory(Bi-LSTM) networks are used to extract information in the `reviewText` column as this column contains much longer text data (up to 1200 words at maximum after cleaning). Bi-LSTM is expected to learn information about the information in the context of each node.

A typical structure of single forward LSTM is shown in the appendix(Figure 2). An input node represents a GloVe transformed vector and is sent into the LSTM cell. By updating the hidden state of the cell, some information is preserved and some useless information is forgotten. A bidirectional LSTM combines two layers of LSTM which have opposite calculation directions.

Some works have proven that using the Bi-LSTM structure could enhance the performance of language models.(Xu et al., 2019)

3.2.3 TextCNN

In 2014, Yoon Kim proposed a structure of the convolutional neural network that outperforms many other models.(Figure 3) TextCNN structure consists of 2 main phases:

1. Convolution phase. As the input is a series of word vectors that are converted by the embedding layer, kernels of the same dimension(width) but with different lengths are used to perform element-wise multiplication to the input matrix.
2. After the activation function, the computed vectors are going through the max-pooling layer to keep the maximum value and then concatenate as the input of the final full connect to network.

By interpreting the structure in tuition, the CNN can be used to extract information and learn useful patterns by combining adjacent words - phrases or small expressions.

3.2.4. Ordinal Regression

In this task, sentiments are not only positive and negative that can be treated as a traditional binary classification problem. The targets are ordered and have multiple classes. If this task is treated as a multi-class classification problem, information about the order is lost and surely will cause performance loss. If this task is treated as a regression problem, class 5 should have 5 times the sentiment of 1. And bias also occurs when thresholds are applied to the output.

Therefore, the method to get the output should be consistent, ordered and unbiased. We are using the method used in age estimation in the image recognition field. The multiple outputs of the neural network represent the probability that the prediction is greater than a certain value of rank. This method has proven to be rank consistent and has better performance than existing classification and regression methods.

3.3 TASK C

3.3.1 BERT

The transformer model, named Bidirectional Encoder Representations from Transformers(BERT), is used in this task to explore the state of art techniques of NLP. When humans interpret the language, they go from start to end. In comparison, Bert transformer goes both directions to help computers understand the text by its surrounding context. Substituting the embedding layer with BERT transformers is expected to enhance the model performance.

4. Experiment details

4.1. Exploratory Data Analysis(EDA)

EDA in this task is relatively simple because there are only two columns of textual data.

Here are some conclusions based on EDA:

1. There are no missing entries

2. The target is a little imbalanced - with the rating 3.0 having relatively fewer entries and 4.0 relatively more.
3. The *reviewText* column is right-skewed and can contain really long text data. The `summary` column is also right-skewed but has much less text.
4. The length of the text columns has little difference among different rating classes.
5. The overlap coefficient between the two also has little relationship to the rating distribution

4.2. Preprocessing

Raw data extracted from the internet usually contains a lot of noise - emoticons, HTML entities, etc. Preprocessing is necessarily needed and will be updated based on revisiting the dataset.

4.2.1. Special text transformation

The preprocessing methods and steps are shown below with examples.

Methods	Before preprocessing	After preprocessing
HTML entity transform	’	’
Contraction expansion	You’re	You are
Emoticon transform	:)	happyface
Adding spaces	**4.5 rating**	** 4.5 rating **
Transform to all lower cases	I am	i am

*Concerning the emoticons, 6 categories of emoticons are established and transformed to the corresponding “xxxxxface”, detailed codes are provided in the appendix.

4.2.2 Noise removal

SpaCy tokenizer and lemmatizer are used, turning the document into a list of words while changing the word to its original form. For example, “lemmatized” to “lemmatize”. The reason for using the spaCy package is that it provides named entity recognition and sentence parsing so that some words will maintain unchanged to keep the information.

Subsequently, custom stop words and punctuations(except ! ? and …) are removed as they contain little information about the sentiment.

Lastly, the *sym_spell* package is used to correct all the misspellings to provide a cleaner, normalized corpus.

4.3 Task A

4.3.1 Splitting train, valid and test dataset

In order to evaluate the model performance, the test set(⅙ of the data) is set aside to evaluate the generalization of the model. Another split is conducted to build a validation set as a hyperparameter that can be tuned by comparing the results of different performances of validation sets. Due to the small amount of data, the train-validation splitting is substituted by 5-fold cross validation. Cross validation is able to train the model with all the data points with equal possibility by iterating several times, which makes the outcome more stable.

Stratified splitting is also enabled as the target is imbalanced. This allows different ratings are equally distributed in train, valid and test sets as it is in the original dataset. Doing so makes sure every class is trained sufficiently, which improves the robustness of outcomes.

4.3.2 TF-IDF Feature extraction

TfidfVectorizer from sklearn is used to build features from the text. Vocabularies that have frequency over 95% or occurrence below 10 times will be ignored. In addition, from the data preprocessing, the maximum length in review is 1270, so setting the maximum of features as 1000, to contain as many vocabularies as possible.

4.3.3 Modelling (gradient boosting and regression)

Using Pipeline from sklearn to combine TF-IDF converting and Gradient Boosting modelling to avoid data leakage. For regression, GradientBoostingRegressor from sklearn will be used, and search for the best parameters for learning_rate, n_estimators, max_depth and subsample by GridSearchCV. The best parameters are learning_rate: 0.1, max_depth: 5, n_estimators: 200, subsample: 0.8.

One alternative way of converting the continuous output is to round the result to the nearest 1, 2, 3, 4, 5 level. This obviously contradicts our EDA where in rating 5 there are more samples. Although the assumption in distribution is not strict, each level of the result should not be too small. Therefore, the threshold here adopts the method of conforming to the previous level's distribution. In other words, the proportion of each level in the training set is calculated and mapped these 'percentiles' into predicted results, to divide them into 5 classes.

It shows that there are 1700 counts for rating 1, 1500 for rating 2, 1200 for rating 3, 2400 for rating 4 and 2200 for rating 5. Using these counts to calculate the proportion, then sort the predicted results from small to large, and split the predictions into classification results. In this way, all numbers are converted into ratings and have a reasonable distribution. Finally, predictions are restored to the original order.

4.3.4 Modelling (gradient boosting and classification)

For classification, GradientBoostingClassifier is used. The best parameters are: learning_rate: 0.1, max_depth: 5, n_estimators: 200 and subsample: 0.8. The model uses the test set to get the predictions.

4.3.5 Results and comparing regression way and classification way

Accuracy_score and f1_score from sklearn are used to evaluate two models while using average='macro' for F1 score. The result is the following table:

	Accuracy	F1 score
Regression	0.4333	0.4143
Classification	0.4820	0.4507

Overall, two ways model has very close performance in Accuracy score and F1 score, but the classification model is slightly better than the regression model. The thresholds used here assume that the prediction should have the same distribution as the target. And ideal thresholds should be the same as 1 to 5, such discrepancy causes much reduction in F1 score.

4.4 TASK B

With the same preprocessing methods, we try to use advanced neural network models in this section to seek a better performing model.

4.4.1 The building blocks and structure of the Bi-LSTM-TextCNN network

Considering `reviewText` data have long texts, the Bi-LSTM block is used after the embedding layer to learn context information around each word vector. Each step's output of LSTM is used as the output to be learnt at the TextCNN block, in order to extract important information in specific areas. `summary` has much shorter words and can be directly learnt through TextCNN.

In the forward propagation view, these two columns are separately learnt because we consider `summary` as a more important indicator of sentiments and `reviewText` serves as a fine-tune factor of the result. These two inputs should not share the same weights in the TextCNN block and are expected to be learnt in the network to have different importance.

The whole structure visualization of Bi-LSTM-TextCNN-Ordinal regression is shown in the appendix.

4.4.2 Output Layer and Loss Function

In this sentiment analysis, we should not simply regard the task as a classification task. We will suffer a loss of information in the order of sentiment rating. Initially, we came up with 2 solutions to this issue. Firstly, we can simply regard the task as a regression task as in Task A, which proves to be less accurate than classification. We also tried to encode the target to a list of new targets which can represent the ordinal information. (Jianlin Cheng, 2008) But this solution has a potential problem that sometimes we can not decode results. For example, if we get a result like [1,0,1,1,1], we can not easily define which group it belongs to.

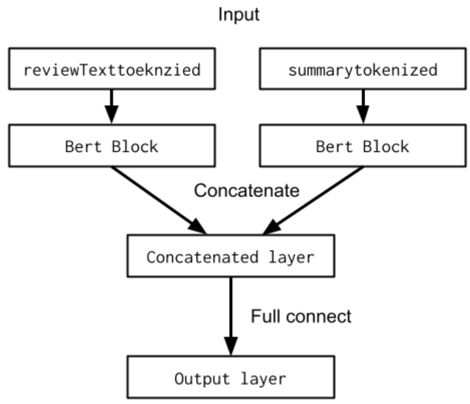
The final solution is using ordinal regression as is proposed in section 3. We tried to make the output layer have multiple outputs. Given that there are 5 categories, the target is one hot encoded, as shown in the structure, interpreted as the probability of the predicted value greater than 1 to 4. The loss function is designed as cross-entropy loss (Cao et al., 2020). For rank prediction, there will be 4 continuous outputs and the output channels will be marked as 1 if their value is greater than 0.5, and 0 if their value is less than 0.5. It is mathematically proved that the output will be rank consistent. The ultimate rank prediction uses the same decoding as one-hot.

The result is that this model will have a much more stable performance than without such a technique - the standard deviation of the validation F1 score is much lower(which is observed in different epochs). However, the maximum value of the validation f1 score is not as large as the network without using this method.

We also conducted several hyperparameter tuning but failed to find the optimized one that can outperform the gradient boosting classification model. (Results are shown in Table 1)

It is considered that the model is too complicated while the dataset is too small so that the model is not able to learn enough samples to have a decent prediction ability. Moreover, it is proposed that TextCNN can be used after embedding to learn short expressions at first and then feed into the Bi-LSTM block to learn contextual information. Due to the limitation of computation power and time, experiments are far from enough to draw a convincing conclusion.

4.5 TASK C



Due to the limitation of computation power, we only considered the simplest structure of implementing the BERT model. Separating `reviewText` and `summary` is still considered as we expected them to have different importance to the model prediction.

As a result, given the learning rate of 2e-5, the BERT model shows an F1 score of 0.3533.

5. Conclusion

5.1. Recommendation

It is recommended that the company that uses this technical report should choose a simpler gradient boosting classification model, the reason is as follows:

1. The computational cost. Gradient boosting has a much higher training and predicting speed as its structure is relatively simple. In some sentiment analysis tasks, low responding speed may affect the user's experience. Moreover, book review data are text data from online as there are some trendy expressions that may change over time, so the company will have to update the model quite frequently, the simpler model can save more time and cost.
2. Model performance. As is compared above, gradient boosting has better performance than neural network based models. The accuracy is the most direct and important factor in choosing a model.
3. Interpretability. As a tree based algorithm, gradient boosting can be understood with the right tool(some tree model visualizations), while neural networks can hardly explain what every neuron has a specific meaning (only could be explained layer-wise).

However, this does not necessarily mean that neural networks should be abandoned. Firstly, the performance gap is not large but the experiments are based on a small dataset, which means the outcome may be different in the actual deployment environment as there are millions of data available. Secondly, there is still a lot of space for optimizing the model - try different structures and complexity - and researches have proven that these advanced neural network models could outperform the traditional ones as the neural network is able to learn semantic information.

5.2. Reflections

Due to the limited time and computational power, we considered several limitations of this study:

1. There is no comparison between the different structures of neural networks in task B, which may improve the model performance. And more hyperparameters can be tuned.
2. Concerning Task A, LightGBM could be a better solution as an advanced gradient boosting method - It has a faster computation speed and better generalization in practice. The ordinal regression method may also work as long as the gradient boosting model can have multiple outputs.
3. We should have dug deeper into the cutting edge techniques of transformer models, there is no comparison using a similar structure to compare the transformer with the embeddings, because the BERT transformation block has already consumed lots of computation power and we have to directly link it to a full connect output layer.
4. As for preprocessing, the data that is learnt by neural networks may not be deep cleaned because the neural network may learn from the propositions or specially named entities. Moreover, the proportion of uppercase in the text should also be considered in the model as they imply a stronger expression of feelings.

6. Reference

Arief, R., Achmad, B. M., Tubagus, M. K., & Hustinawaty, H. (2022). Automated hierarchical classification of scanned documents using convolutional neural network and regular expression. *International Journal of Electrical and Computer Engineering*, 12(1), 1018-1029. https://doi.org/10.11591/ijece.v12i1.pp1018-1029

Asghar, M. Z., Khan, A., Ahmad, S., & Kundi, F. M. (2014). A review of feature extraction in sentiment analysis. *Journal of Basic and Applied Scientific Research*, 4(3), 181-186.

Cao, W., Mirjalili, V., & Raschka, S. (2020). Rank consistent ordinal regression for neural networks with application to age estimation. *Pattern Recognition Letters*, *140*, 325-331. https://doi.org/10.1016/j.patrec.2020.11.008

Chen, L., Lee, C. & Chen, M. (2020). Exploration of social media for sentiment analysis using deep learning. *Soft Computing*, 24(11), 8187-8197. https://doi.org/10.1007/s00500-019-04402-8

Chen, T., Xu, R., He, Y., & Wang, X. (2017). Improving sentiment analysis via sentence type classification using BiLSTM-CRF and CNN. *Expert Systems With Applications*, *72*, 221-230. https://doi.org/10.1016/j.eswa.2016.10.065

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* preprint arXiv:1810.04805.

Fersini, E., Pozzi, F. A., & Messina, E. (2017). Approval network: a novel approach for sentiment analysis in social networks. *World Wide Web*, 20(4), 831-854. https://doi.org/10.1007/s11280-016-0419-8

Imane, G., Kareem, D., & Faical, A. (2019). A set of parameters for automatically annotating a Sentiment Arabic Corpus. *International Journal of Web Information Systems*, 15(5), 594-615. https://doi.org/10.1108/IJWIS-03-2019-0008

Jianlin Cheng; Zheng Wang; Gianluca Pollastri, A neural network approach to ordinal regression (2008), *IEEE International Joint Conference on Neural Networks*

Kamyab, M., Liu, G., & Adjeisah, M. (2021). Attention-Based CNN and Bi-LSTM Model Based on TF-IDF and GloVe Word Embedding for Sentiment Analysis. *Applied Sciences*, 11(23), 11255. https://doi.org/10.3390/app112311255

Liu, C., & Xu, Y. (2021). Consumer Sentiment Involvement in Big Data Analytics and Its Impact on Product Design Innovation. *Sustainability*, 13(21), 11821. https://doi.org/10.3390/su132111821

Mäntylä, M., Graziotin, D., & Kuutila, M. (2018). The evolution of sentiment analysis—A review of research topics, venues, and top cited papers. *Computer Science Review*, *27*, 16-32. https://doi.org/10.1016/j.cosrev.2017.10.002

Moghadasi, M., & Zhuang, Y. (2020). Sent2Vec: A New Sentence Embedding Representation With Sentimental Semantic. *2020 IEEE International Conference On Big Data (Big Data)*. https://doi.org/10.1109/bigdata50022.2020.9378337

Nasukawa, T., & Yi, J. (2003). Sentiment analysis Capturing favourability using Natual Language Processing. *Proceedings Of The International Conference On Knowledge Capture - K-CAP '03*. https://doi.org/10.1145/945645.945658

Ng, C. Y., Law, K. M. Y., & Ip, A. W. H. (2021). Assessing Public Opinions of Products Through Sentiment Analysis: Product Satisfaction Assessment by Sentiment Analysis. *Journal of Organizational and End User Computing*, 33(4), 125. https://doi.org/10.4018/JOEUC.20210701.oa6

Saad, S., & Yang, J. (2019). Twitter Sentiment Analysis Based on Ordinal Regression. *IEEE Access*, *7*, 163677-163685. https://doi.org/10.1109/access.2019.2952127

Sangeetha, K., & Prabha, D. (2021). Sentiment analysis of student feedback using multi-head attention fusion model of word and context embedding for LSTM.*Journal of Ambient Intelligence and Humanized Computing*, 12(3), 4117-4126. https://doi.org/10.1007/s12652-020-01791-9

Srujan, K., Nikhil, S., Raghav Rao, H., Karthik, K., Harish, B., & Keerthi Kumar, H. (2018). Classification of Amazon Book Reviews Based on Sentiment Analysis. *Advances In Intelligent Systems And Computing*, 401-411. https://doi.org/10.1007/978-981-10-7512-4_40

Tor Ole, B. O., Marin, A., & Rudolph, J. L. (2021). How has 0RW1S34RfeSDcfkexd09rT2Science Education1RW1S34RfeSDcfkexd09rT2 changed over the last 100 years? An analysis using natural language processing. *Science Education*, 105(4), 653-680. https://doi.org/10.1002/sce.21623

Wenzhi Cao, Vahid Mirjalili & Sebastian Raschka(2020). Rank consistent ordinal regression for neural networks with application to age estimation. *Pattern Recognition Letters*, , 325-331, 140. https://doi.org/10.1016/j.patrec.2020.11.008

Wu, S., Li, J., Wang, T., & Chen, Z. (2019). Quality Evaluation System of Online Courses Based on Review Data. *International Journal of Design, Analysis and Tools for Integrated Circuits and Systems*, 8(1), 19-23. http://ezproxy.library.usyd.edu.au/login?url=https://www.proquest.com/scholarly-journals/quality-evaluation-system-online-courses-based-on/docview/2316725529/se-2?accountid=14757

Xu, G., Meng, Y., Qiu, X., Yu, Z., & Wu, X. (2019). Sentiment Analysis of Comment Texts Based on BiLSTM. *IEEE Access*, *7*, 51522-51532. https://doi.org/10.1109/access.2019.2909919

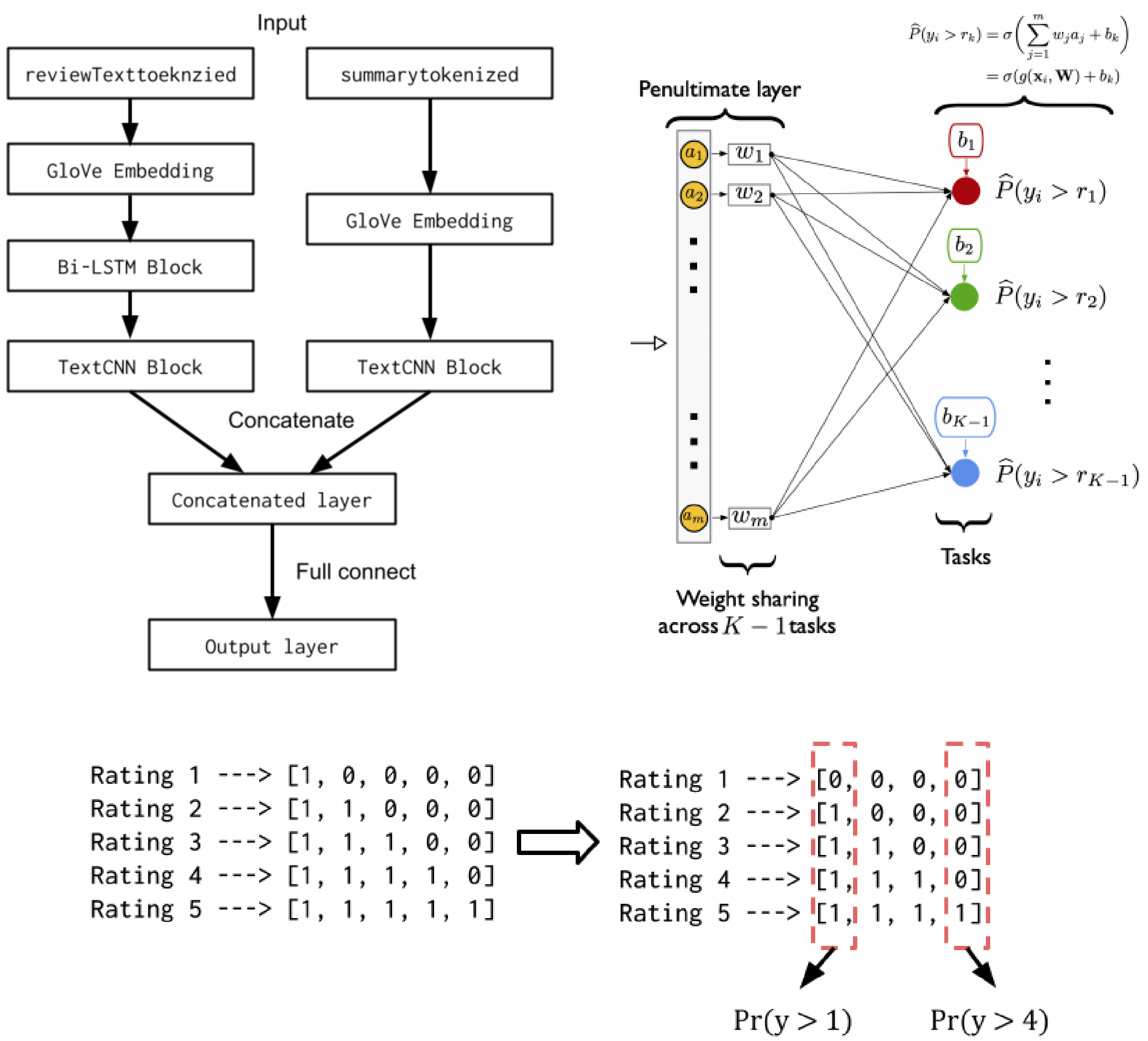
Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1746–1751, Doha, Qatar. *Association for Computational Linguistics*.https://doi.org/10.48550/arXiv.1408.5882

Zhang, S. (2021). Sentiment analysis based on food e-commerce reviews. *IOP Conference Series.Earth and Environmental Science*, 792(1)https://doi.org/10.1088/1755-1315/792/1/012023

Appendix

Charts and Graphs

Figure 1: The structure of Task B model



Up-left: the structure of the whole network;

Up-right: the encoding methods for ordinal regression

Downside: The output layer design)

Figure 2 The basic node structure of LSTM

image source https://www.jiqizhixin.com/articles/2018-10-24-13

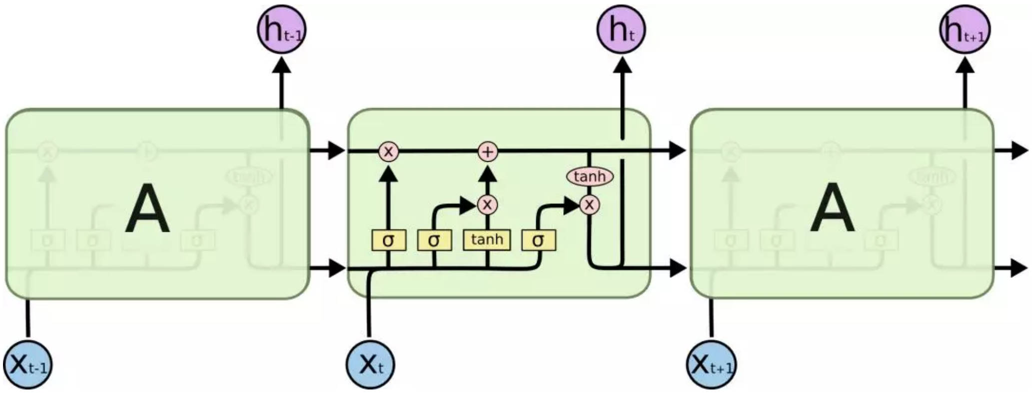


Figure 3 The Basic structure of TextCNN

image source：https://zhuanlan.zhihu.com/p/77634533

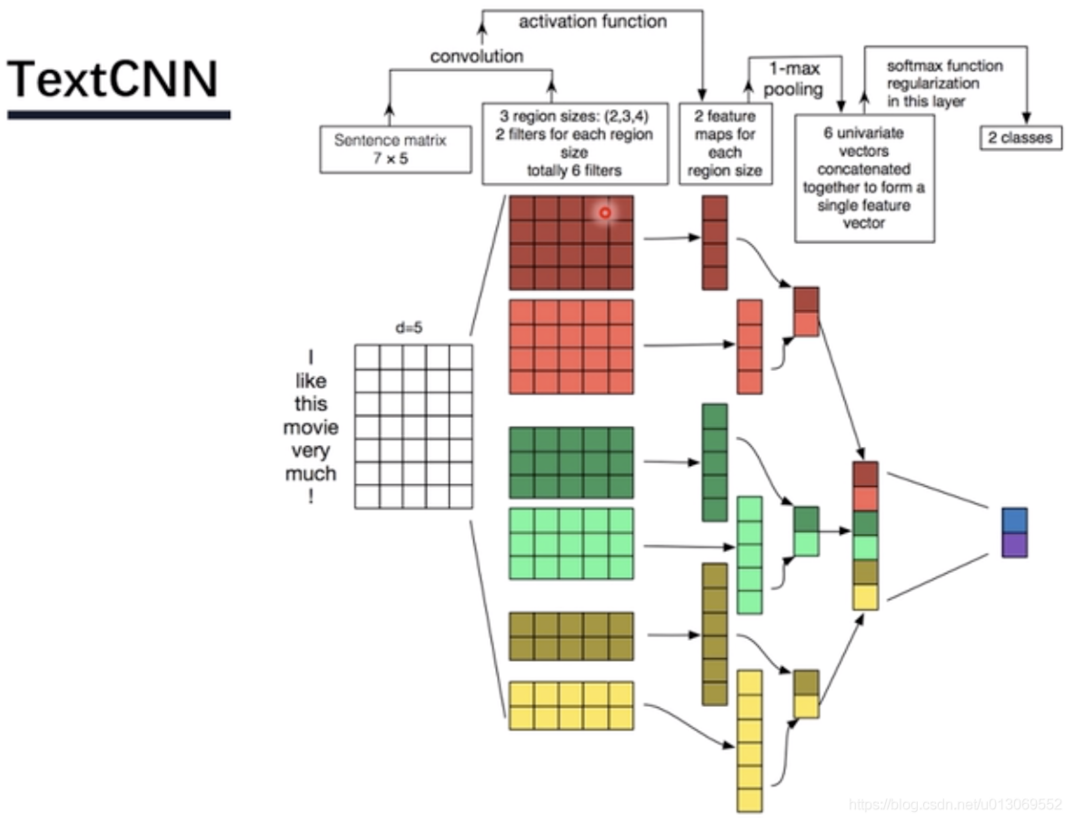


Table 1 The Result of Hyperparameter optimization in Task B

parameter	base	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10	test11	test12
batch_size	128	-	-	-	-	-	-	-	-	-	-	-	-
epoch	12	-	-	-	-	-	-	-	-	-	-	-	-
hidden_size	100	-	-	-	-	-	-	-	-	-	-	200	50
dropout	0.1	-	-	-	-	-	0.2	-	-	-	-	-	-
lstm_layers	1	-	-	-	-	2	-	-	-	-	-	-	-
optimizer learning rate	0.001	-	-	-	-	-	-	0.0005	-	-	-	-	-
weight_decay	0.001	-	-	-	-	-	-	-	0.005	0.0005	-	-	-
kernel_num	32	-	-	16	64	-	-	-	-	-	-	-	-
kernel_size	2,3,4	-	-	-	-	-	-	-	-	-	2,4,6	-	-
upper count	false	true	-	-	-	-	-	-	-	-	-	-	-
ordinal regression	True	-	False	-	-	-	-	-		-	-	-	-
result	0.402	0.426	**0.443**	0.438	0.440	0.372	0.442	0.431	0.409	0.426	0.423	0.431	0.426

*Upper count - is whether consider the upper case as another input in the neural network.

Duty Declaration

SID	TASKS
480397865	Introduction, Literature review, Presentation management
510160513	Task A Description & discussion of the models
500322217	Task A Experiment details & result analysis
510170981	Preprocessing, Task B Abstract Conclusion Model/Method optimisation
510207449	Task B & C Kaggle competition Model/Method optimisation

Code

EDA & Preprocessing

```
#!/usr/bin/env python
# coding: utf-8

# ## 1. Importing Libs / Data loading / Environment setting

# In[1]:


# basic lib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
get_ipython().run_line_magic('matplotlib', 'inline')
import missingno as msn
import warnings
import random
import math

# setting
warnings.filterwarnings('ignore')
pd.set_option("display.max_columns", 20)


# In[2]:


from sklearn.model_selection import train_test_split
import re

import torch

# reproducibility (global setting)
torch.manual_seed(12)
np.random.seed(12)
random.seed(12)

import pkg_resources
from symspellpy import SymSpell, Verbosity

from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
import contractions
import html
import spacy
#spacy.cli.download("en_core_web_sm")
spacyNLP = spacy.load("en_core_web_sm")


# In[3]:
```

```
# read data
originData = pd.read_csv("train.csv")

# create a deep copy - make sure the original dataset is not changed.
dataset = originData.copy(deep = True)


# ## 2. EDA
#

# In[4]:


dataset.sample(10)


# - `rating` - the target
# - `reviewText` - long review, feature
# - `summary` - short review, feature

# ### 2.1. missing values

# In[5]:


# missing value
msn.bar(dataset, color='lightsteelblue')


# no missing value

# ### 2.2. Target

# In[6]:


sns.histplot(data = dataset, x = "rating", bins = 5)


# Several conclusions:
# - This is a multi-classfication problem.
# - Classes have order - Ordinal classification.
# - The target is slightly umbalanced, 4 and 5 has more examples and 3
has less examples.

# ### 2.3 Text Features

# In[7]:


# reviewText
dataset['reviewTextCharCount'] = dataset['reviewText'].apply(len)
sns.histplot(dataset['reviewTextCharCount'], bins = 100)


# There are some reviews that writes a really long article while some
are just a few words.
#
```

```
# Apparently right skewed.

# In[8]:


# summary
dataset['summaryCharCount'] = dataset['summary'].apply(len)
sns.histplot(dataset['summaryCharCount'], bins = 12)


# `summary` has much shorter length but it is also right skewed.

# In[9]:


fig, (ax1, ax2) = plt.subplots(1,2,figsize = (8,4))
sns.boxplot(x = 'rating',y='summaryCharCount', data = dataset,
showfliers = False, ax= ax1)
sns.boxplot(x = 'rating',y='reviewTextCharCount', data = dataset,
showfliers = False, ax = ax2)
plt.subplots_adjust(wspace=0.4)
plt.show()


# As is shown in the boxplot(outlier excluded), there is little
difference in the length of reviews and summaries between different
ratings.
#
# Review text length will be less on rating 1 and 5, with other
ratings have longer text, the difference is small though.
#
# So merely the length of the text will not be a good indicator of
rating.

# ### 2.4. Between `reviewText` and `summary`
#

# In[10]:


# define the overlap similarity score
# we use overlap instead of other methods
# because the difference in length between two columns are quite big
def overlapSim(str1, str2):
    a = set(str1.lower().split())
    b = set(str2.lower().split())
    c = a.intersection(b)
    return float(len(c)) / min(len(a),len(b))

overlapScoreResult = []
for i,row in dataset.iterrows():
    s1 = row.reviewText
    s2 = row.summary
    overlapScore = overlapSim(s1, s2)
    overlapScoreResult.append(overlapScore)

dataset.insert(loc = 4,
              column = 'overlapScore',
```

```
                value = overlapScoreResult)


# In[11]:


sns.histplot(dataset['overlapScore'],bins = 10)


# In[12]:


sns.boxplot(x = 'rating',y='overlapScore', data = dataset)


# There is a large proportion of reviews that its summary has no **word similarity** with its content.
#
# However, this does not necessarily mean they do not have **semantic similarity**, more advanced techniques(using trained model / vectorized representation / after normalization) can be used to analyse the sementical similarity.
#
# The reason why similarity is analysed here is that `summary` may not provide accurate attitude and will be elaborate on the longer `reviewText`.

# ## 3. Data Preprocessing
# We now have only the text data, which should be transformed into numerical representation to feed the model as computer can only handle structured data.
#
# Before that, texts should be normalized to reduce the variation:
#
# Example:
#
# `'good', 'GOOD', 'Goooood', 'best', 'great', 'graet', 'good!'` should mean the same thing with only slight difference in terms of **form, puntuation, upper/lower case, typo, etc.**. But the model may consider them as different word if only simple methods is used to transform them.
#
# For better generalization. Steps including:
# - lower case
# - contraction expansion
# - stop word removal
# - punctuation removal
# - lemmatization and stemming
# - part of speech tagging
# - ...
#
# **NOTE** that data quality is the most important thing - GARBAGE IN, GARBAGE OUT - The normalization steps should not have too much information loss.
#
# There are many methods to tranform the texts into numbers:
# 1. Bag of words
# 2. TF-idf
```

```
# 3. Advanced embedding method (vectorization)
#
# **Note that some methods may cause data leakage** - methods that
involve information in the test/validation set. - We should retrict
these methods to be deployed in the training set only.

#

# In[13]:


# convert html entities
# &#8217; => '

def htmlEntityTran(DF, columns):
    for acol in columns:
        DF[acol] = DF[acol].apply(lambda x: html.unescape(x))


# In[14]:


# expand contractions such as
# I'm -> I am

def contractionExpansion(DF, columns):
    for acol in columns:
        DF[acol] = DF[acol].apply(lambda x: contractions.fix(x))


# In[15]:


def punctuationProcess(DF, columns):
    for acol in columns:
        DF[acol] = DF[acol].apply(lambda x: puncFix(x))

smileEmo = r""":-) :) :-] :] :-> :> 8-) 8) :-} :} :o) :c) :^) =] =)
^_^ ^^ :') :3 :-3 =3 x3 X3 (: (-: ))""".split(" ")

laughEmo =  r""":-D :D 8-D 8D =D =3 B^D c: x-D xD X-D XD
C:]""".split(" ")

winkEmo = r""";-) ;) *-) *) ;-] ;] ;^) ;> :-, ;D ;3""".split(" ")

sadEmo = r""":-( :( :-c :c :-< :< :-[ :[ :-|| >:[ :{ :@ :( ;( :'-( :'(
:=( :$ ): """.split(" ")

skepEmo =r"""
:-/
:/
:-.
>:\
>:/
=/
=\
:L
=L
```

```
:S
:-|
:|
-_-
""".split()

stunEmo = r""":-O :O :-o :o :-0 8-0 >:O =O =o =0 O_O o_o O-O o-o O_o
o_O""".split(" ")

def puncFix(row):
    for emoticon in smileEmo:
        row = row.replace(emoticon, 'smileface')
    for emoticon in laughEmo:
        row = row.replace(emoticon, 'laughface')
    for emoticon in winkEmo:
        row = row.replace(emoticon, 'winkface')
    for emoticon in sadEmo:
        row = row.replace(emoticon, 'sadface')
    for emoticon in skepEmo:
        row = row.replace(emoticon, 'skepticalface')
    for emoticon in stunEmo:
        row = row.replace(emoticon, 'stunnedface')

    row = row.replace('$$','ss')
    row = row.replace('$','money')
    row = row.replace('w/','with')
    row = re.sub(r"""(?<=[,.!'";:()*?/-])(?=[a-zA-Z])""", ' ', row)
    row = re.sub(r"""(?<=[a-zA-Z])(?=[,.!'";:()*?/-])""", ' ', row)
    row = re.sub(r"[...][.]+", '...', row)
    row = re.sub(r"[--][-]+", '--', row)
    row : row.replace('**', ' ** ')
    return row


# In[16]:


# lower case
# A => a
def lowercaseCountTranformer(DF, columns):
    """
    This function transform all the characters into lower case,
    And store the number of upper case
    """
    for acol in columns:
        DF[acol+'UpperCount'] = DF[acol].apply(lambda x: sum(1 for c
in x if c.isupper() ) )
        DF[acol] = DF[acol].apply(lambda x: x.lower())


# In[17]:


# lemmatization and stemming
# turn the word back to its original form

# tokenization
# turning a sentense into a list of words
```

```
def LemmatizationTransform(DF, columns, mode = "Lemma"):
    """
    spacy is used but only the tokenization and lemmatization pipeline
component is implemented on the data.
    NLTK has some similar function but lack the precision

    Example:
    'lemmatizing'
    in nltk -> lemmatiz
    in spacy -> lemmatize

    tokenization is also done by this step together in SpaCy
    """
    for acol in columns:
        if mode == "Lemma":
            DF[acol+'tokenized'] = DF[acol].apply(lambda row:
[token.lemma_ for token in spacyNLP(row)])
        else:
            DF[acol+'tokenized'] = DF[acol].apply(lambda row:
[token.text for token in spacyNLP(row)])


# In[18]:


print(stopwords.words('english'))


# In[19]:


origin = set(stopwords.words('english'))

wanted =
{'what','but','against','down','up','on','off','over','under','out','s
ame'
                'again','further','why','what','how', 'all',
'any','with'
                'few', 'more', 'most', 'other','no', 'nor', 'not',
'only',
                'than', 'too', 'very', 'just', 'should', 'ain',
                'aren', "aren't", 'couldn', "couldn't", 'didn',
"didn't",
                'doesn', "doesn't", 'hadn', "hadn't", 'hasn',
"hasn't", 'haven',
                "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't",
'mustn',
                "mustn't", 'needn', "needn't", 'shan', "shan't",
'shouldn', "shouldn't",
                'wasn', "wasn't", 'weren', "weren't", 'won', "won't",
'wouldn', "wouldn't"}

unwanted = {'the','I',"'s"}

StopWordCustom_Deep = list(origin - wanted | unwanted)

StopWordCustom_Shallow = ["it's", 'their', 're', 'she', 'ours',
```

```
                          'it', 've', 'you', 'y', 'o', 'themselves',
                          'your', 'yours', 'm', "you'd",
                          "you're", 'and', 'its', "you've", 'that',
'ourselves',
                          'himself', 'this', 'been', "you'll", 'an',
'my', 'me',
                          'myself', 'a', 'these', 'which',
                          'he', 'his', 'I', 'them', 'the', "'s",
'yourselves',
                          'our', 's', 'yourself', 'theirs', 'herself',
'they', "she's",
                          'hers', 'we', 'those', 'him', "that'll",
'i', 'her', 'itself']
# stopword removal
def stopWordRemove(DF, columns, deep = True):
    for acol in columns:
        if deep == True:
            DF[acol] = DF[acol].apply(lambda alist: [item for item in
alist if item not in StopWordCustom_Deep])
        else:
            DF[acol] = DF[acol].apply(lambda alist: [item for item in
alist if item not in StopWordCustom_Shallow])


# In[20]:


# punctuation
# '...' and '!' and '?' would contain some information about the
sentiment

punctList = set(string.punctuation)
puncwanted = {'!', '?'}
puncunwanted = {' '}
punctList = punctList - puncwanted | puncunwanted

def punctRemover(DF, columns):
    for acol in columns:
        DF[acol] = DF[acol].apply(lambda alist: [item for item in
alist if item not in punctList])



# In[21]:


punctuationException = {'?','!','...'}
others = {'eh'}
symspellException = punctuationException | others


# In[22]:


sym_spell = SymSpell(max_dictionary_edit_distance=2, prefix_length=7)
dictionary_path = pkg_resources.resource_filename(
    "symspellpy", "frequency_dictionary_en_82_765.txt"
)
```

```
# term_index is the column of the term and count_index is the
# column of the term frequency
sym_spell.load_dictionary(dictionary_path, term_index=0,
count_index=1)

def spellcheck(tokens):
    checkedtokens = []
    for token in tokens:
        if (token not in symspellException) and (not
token.isnumeric()):
            try:
                checkedtokens.append(sym_spell.lookup(token,
Verbosity.CLOSEST, max_edit_distance=2)[0].term)
            except:
                checkedtokens.append(token)
        else:
            checkedtokens.append(token)
    return checkedtokens

def spellCheckReplace(DF, columns):
    for acol in columns:
        DF[acol] = DF[acol].apply(spellcheck)


# - The html entity code for special characteristics: **(solved)**
#
# iloc[2941] - `&#34;`
#
# iloc[7085] - `&eacute;`
#
# - Some emoji/ simple face expression **(solved)**
#
# > spaCy have the ability in tokenizing the text-based emoticon
#
https://github.com/explosion/spaCy/blob/master/spacy/lang/tokenizer_ex
ceptions.py#L115
#
# iloc[3697] - `:)`
#
# iloc[8983] - `;-)`
#
# - repeated punctuation (not solved)
#
# iloc[2998] - `WHAT?!?!?!?!`
#
# iloc[7486] - `...`
#
# - Abbreviation (not solved)
#
# iloc[5884] - `def.` == `definitely`
#
# iloc[8503] summ. - `$` == money
#
# iloc[8163] summ. `w/` == with
#
# - Direct information (not solved)
#
# iloc[4695] - `**4.5 stars**`
```

```
#
# - Wierd expressions (not solved)
#
# iloc[2873] - `a$$`
#
# - Typo **solved**
#
# iloc[4026] summ. - `lovrd it`
#
# - `spacy` tokenization problems **solved**
#
# iloc[4695] - `**4.5 stars**` ==> `stars**i`

# More can be done
# - phrase extraction
#     - by adding exceptions in tokenization
# - expansion with synonyms
#     - augmentation
# - dependence parsing
#
# - sentense tokenization
#
# Possibility in improvements in normalization and tokenization:
# - **lower case**, some people may use all upper case to express a
strong feeling and that may be a good indicator of an extreme rating.
# - **punctuation**, some punctuation,  `!` - implies a strong
expression, `?` implies a questioning attitude, `...` implies
skeptical or reserved feelings - decided by context. And some more
advanced expressions like `:)`,`:P`, could also contain some sementic
meanings but is excluded in the analysis.
# - **stop words**, we may add more stop words other than common
one(which should be task-specific) this could be done by looking at
the prediction faeture importance using `SHAP` library.
# - **abbreviation&slang**, there are some abbreviation or slang that
can be extended to its original form that can be further normalize the
dataset.
# These aspects can shed light on the model performance improvement.

# In[ ]:





# In[23]:


# Workflow of deep preprocess
def DeepPreprocess(df):
    htmlEntityTran(df,['reviewText','summary']) # &#34; => "
    contractionExpansion(df,['reviewText','summary']) # I'm => I am
    punctuationProcess(df,['reviewText','summary']) # add space before
puncs; emoticon; normalize
    lowercaseCountTranformer(df, ['reviewText','summary']) # A => a ;
add count column
    LemmatizationTransform(df,['reviewText','summary']) # smiled =>
smile; tokenized
```

```
    stopWordRemove(df,['reviewTexttokenized','summarytokenized']) #
delete "I"
    punctRemover(df,['reviewTexttokenized','summarytokenized']) #
delete meaningless punctuation
    spellCheckReplace(df,['reviewTexttokenized','summarytokenized']) #
graet => great

DeepPreprocess(dataset)


# In[24]:


#DeepPreprocess(dataset)
#subData = pd.read_csv('test.csv')
#DeepPreprocess(subData)
#subData.to_csv('preprocessedtest_deep.csv')
#dataset.to_csv('preprocessedtrain_deep.csv')


# In[26]:


# shallow preprocessing (for neural networks that can learn grammar)
def ShallowPreprocess(df):
    htmlEntityTran(df,['reviewText','summary']) # &#34; => "
    contractionExpansion(df,['reviewText','summary']) # I'm => I am
    punctuationProcess(df,['reviewText','summary']) # add space before
puncs; emoticon; normalize
    lowercaseCountTranformer(df, ['reviewText','summary']) # A => a ;
add count column
    LemmatizationTransform(df,['reviewText','summary'], mode =
"shallow") # only tokenized
    stopWordRemove(df,['reviewTexttokenized','summarytokenized'], deep
= False) # delete "I"
    punctRemover(df,['reviewTexttokenized','summarytokenized']) #
delete meaningless punctuation



# In[27]:


# run this to do shallow preprocess
# keep the proposition
# avoid using misspelling correction -
# it may do wrongly and convert useful information to something else

#subData = pd.read_csv('test.csv')
#ShallowPreprocess(dataset)
#ShallowPreprocess(subData)

subData.to_csv('preprocessedtest_deep.csv')
dataset.to_csv('preprocessedtrain_deep.csv')


# ## Data quality checker
```

```
# In[28]:


# data quality checker
a = random.randint(0,9000)
a = 8983
print(a)
print(dataset['reviewTexttokenized'].iloc[a])
print("===============================")
print(dataset['summarytokenized'].iloc[a])
print("===============================")
print(dataset['reviewText'].iloc[a])
print("===============================")
print(dataset['summary'].iloc[a])
print("*****************************")
print(originData['reviewText'].iloc[a])
print("===============================")
print(originData['summary'].iloc[a])


# In[29]:


from collections import Counter
counter = Counter()

MaxLengthReview = 0
MaxLengthSummary = 0

for row in dataset['reviewTexttokenized']:
    counter.update(row)
    if len(row) > MaxLengthReview:
        MaxLengthReview = len(row)

for row in dataset['summarytokenized']:
    counter.update(row)
    if len(row) > MaxLengthSummary:
        MaxLengthSummary = len(row)

print('MaxLengthReview', MaxLengthReview)
print('MaxLengthSummary',MaxLengthSummary)


# In[30]:


for akey in sorted(counter.keys()):
    for char in akey:
        a = 0
        if char in string.punctuation:
            a = a + 1
    if a > 0:
#        if counter[akey]>=2:
        print(akey, '|===>', counter[akey])
```

TASK A

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


# basic lib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
get_ipython().run_line_magic('matplotlib', 'inline')
import missingno as msn
import warnings
import random
import math

# setting
warnings.filterwarnings('ignore')
pd.set_option("display.max_columns", 20)


# In[2]:


from sklearn.model_selection import train_test_split
import re


# In[3]:


dataset = pd.read_csv('preprocessedtrain_deep.csv',
                      index_col = 0,
                      converters = {'reviewTexttokenized': eval,
                                    'summarytokenized': eval}
                     )


# In[4]:


dataset.sample(3)


# In[5]:


dataset['reviewCapitalPer'] = dataset['reviewTextUpperCount'] /
dataset['reviewTextCharCount']
dataset['summaryCapitalPer'] = dataset['summaryUpperCount'] /
dataset['summaryCharCount']


# ## TF-iDF
```

```
# In[6]:


from sklearn.feature_extraction.text import TfidfVectorizer


# In[7]:


dataset['concattoken'] = dataset.reviewTexttokenized.apply(lambda x:x)
+ dataset.summarytokenized.apply(lambda x:x)


# In[23]:


from sklearn.model_selection import train_test_split, StratifiedKFold

from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.model_selection import GridSearchCV

allX = dataset['concattoken']# ,'reviewCapitalPer','summaryCapitalPer'
ally = dataset['rating']
# split into train and test set
X_tr_va, X_test, y_tr_va, y_test = train_test_split(allX, ally,
                                                    test_size=1/6,
                                                    random_state = 12,
                                                    stratify = ally)

#X_tr_va.reset_index(drop = True, inplace = True)
#X_test.reset_index(drop = True, inplace = True)
#y_tr_va.reset_index(drop = True, inplace = True)
#y_test.reset_index(drop = True, inplace = True)

SKF = StratifiedKFold(n_splits = 5, random_state = 12, shuffle  =
True)
TFVec = TfidfVectorizer(tokenizer=lambda x:x,
                        lowercase = False,
                        max_df = 0.95,
                        min_df = 10,
                        max_features = 1000,
                        ngram_range = (1,1))

my_pipeline = Pipeline([('vectorizer', TFVec),
                        ('GBR', GradientBoostingRegressor())
                       ])

searching_params = {
    'GBR__learning_rate': [0.01, 0.1, 0.2],
    'GBR__n_estimators' : [20, 50, 100, 200],
    'GBR__max_depth' : [3, 5, 8],
    'GBR__subsample' : [0.3, 0.5, 0.8]
}

grid_search = GridSearchCV(my_pipeline, param_grid=searching_params,
                           cv=SKF, n_jobs=-1,
scoring='neg_mean_squared_error')
grid_search.fit(X_tr_va, y_tr_va)
```

```
print(grid_search.best_params_)


# In[24]:


pred_y = grid_search.best_estimator_.predict(X_test)
pred_y = pd.DataFrame(data=pred_y)
pred_y.columns=['pred_y']
dataset['rating'].value_counts()
w1 = 1700/9000
w2 = 1500/9000
w3 = 1200/9000
w4 = 2400/9000
w5 = 2200/9000

# calculate the percentile as threshold
threshold1 = w1
threshold2 = w1+w2
threshold3 = w1+w2+w3
threshold4 = w1+w2+w3+w4
print(threshold1,threshold2,threshold3,threshold4)


# In[25]:


pred_y_sort = pred_y.sort_values(by='pred_y')
t1 = np.percentile(pred_y_sort,threshold1*100)
t2 = np.percentile(pred_y_sort,threshold2*100)
t3 = np.percentile(pred_y_sort,threshold3*100)
t4 = np.percentile(pred_y_sort,threshold4*100)
print(t1,t2,t3,t4)

def cate(x):
    if x <= t1:
        return 1
    elif x > t1 and x <= t2:
        return 2
    elif x > t2 and x <= t3:
        return 3
    elif x > t3 and x <= t4:
        return 4
    elif x > t4:
        return 5

pred_y['pred_y_cate'] = pred_y['pred_y'].apply(cate)


# In[27]:


y_test.reset_index(drop = True, inplace = True)
test_y = pd.DataFrame(data=y_test)
test_y.columns =['test_y']

est_reg = pd.concat([pred_y,test_y],axis=1)
est_reg
```

```
# ## classification

# In[29]:


from sklearn.ensemble import GradientBoostingClassifier


# In[30]:


my_pipeline_cla = Pipeline([('vectorizer', TFVec),
                            ('GBC', GradientBoostingClassifier())
                           ])

searching_params = {
    'GBC__learning_rate': [0.05, 0.1, 0.2],
    'GBC__n_estimators' : [20, 50, 100, 200, 400],
    'GBC__max_depth' : [3, 5, 8],
    'GBC__subsample' : [0.3, 0.5, 0.8]
}

grid_search_cla = GridSearchCV(my_pipeline_cla,
param_grid=searching_params,
                               cv=SKF, n_jobs=-1,
scoring='f1_weighted')
grid_search_cla.fit(X_tr_va, y_tr_va)
print(grid_search.best_params_)


# In[32]:


pred_y= grid_search_cla.best_estimator_.predict(X_test)
pred_y = pd.DataFrame(data=pred_y)
pred_y.columns=['pred_y']
est_cla = pd.concat([pred_y,test_y],axis=1)
est_cla


# ## Model Evaluation

# In[33]:


from sklearn.metrics import accuracy_score,f1_score

columns=[ 'Accuracy', 'F1 socre']
rows=['Regression', 'Classification']
results=pd.DataFrame(0.0, columns=columns, index=rows)

results.iloc[0,0] = accuracy_score(est_reg['test_y'],
est_reg['pred_y_cate'])
results.iloc[0,1] = f1_score(est_reg['test_y'],
est_reg['pred_y_cate'], average = 'macro')
```

```
results.iloc[1,0] = accuracy_score(est_cla['test_y'],
est_cla['pred_y'])
results.iloc[1,1] = f1_score(est_cla['test_y'], est_cla['pred_y'],
average = 'macro')

results.round(4)


# In[71]:


subdataset = pd.read_csv('preprocessedtest_deep.csv',
                         index_col = 0,
                         converters = {'reviewTexttokenized': eval,
                                       'summarytokenized': eval}
                     )
subdataset['concattoken'] =
subdataset.reviewTexttokenized.apply(lambda x:x) +
subdataset.summarytokenized.apply(lambda x:x)

sub_X = subdataset['concattoken']#
,'reviewCapitalPer','summaryCapitalPer'

X_train_tran = TFVec.fit_transform(allX)
X_sub_tran = TFVec.transform(sub_X)
X_sub_tran = X_sub_tran.toarray()
#X_sub_tran = pd.DataFrame(data = X_sub_tran, columns = rs_names)

FinalModel = GradientBoostingClassifier(learning_rate=0.05,
max_depth=8,
                                        n_estimators=400,
subsample=0.3)

FinalModel.fit(X_train_tran, ally)

y_sub = FinalModel.predict(X_sub_tran)
outputdf = pd.DataFrame(data=y_sub)
outputdf.columns=['Prediction']
outputdf.to_csv('submission_GBreg.csv')
```

TASK B

```
#!/usr/bin/env python
# coding: utf-8

# # Task B

# In[1]:


import os
import pandas as pd
import numpy as np
import torch
import warnings
import random
```

```
import math
import time

# setting
warnings.filterwarnings('ignore')
pd.set_option("display.max_columns", 20)

# ===================================

from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import LabelEncoder


# In[2]:


import torch.nn as nn
import torch.nn.functional as F
from collections import Counter
from torchtext import datasets
from torchtext.vocab import GloVe
from torchtext.vocab import vocab

device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
print(device)


# In[3]:


from torch.utils.data import Dataset
from torch.utils.data import DataLoader


# In[4]:


# reproducibility (global setting)

def seed_everything(seed=12):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

seed_everything()


# ### load data

# In[5]:


dataset = pd.read_csv('preprocessedtrain_deep.csv',
```

```
                      index_col = 0,
                      converters = {'reviewTexttokenized': eval,
                                    'summarytokenized': eval}
                     )


# In[6]:


dataset.head(3)


# In[7]:


counter = Counter()

MaxLengthReview = 0
MaxLengthSummary = 0

for row in dataset['reviewTexttokenized']:
    counter.update(row)
    if len(row) > MaxLengthReview:
        MaxLengthReview = len(row)

for row in dataset['summarytokenized']:
    counter.update(row)
    if len(row) > MaxLengthSummary:
        MaxLengthSummary = len(row)

print('MaxLengthReview', MaxLengthReview)
print('MaxLengthSummary',MaxLengthSummary)


# In[8]:


GLovevectors = GloVe(name='6B',
                     dim=200,
                     cache='D:/program files/jupyter
notebook/usyd/6850/cache')


# In[9]:


# all the config in the model and training
class Config():

    #embedding config
       #how many number of words in glove embedding dict
       #if error occurs - change 400001 to 400000
    embed_vocab_num = 400001

    embed_dim = 200 # dimension of the embedding
    embed_trainable = False # whether train(fine tune) the weight of
embedding
```

```
    #Bi-LSTM config
    hidden_size = 100
    output_size = 1
    dropout = 0.1
    lstm_layers = 1

    # CNN config
    kernel_num = 32 # number of kernels
    kernel_size = [2,3,4] # CNN filter size - similar to n-gram

    max_seq_len_review = MaxLengthReview
    max_seq_len_summary = MaxLengthSummary

    batch_size = 128
    epoch = 12

    learning_rate = 0.05

config = Config()


# In[10]:


def Token2EmbedIndex(row):
    """
    replace the token with the index in the glove dictionary
    if the word cannot be located in the dictionary, it will be
assigned
    with the same index as <unk> - the last key in the dictionary
    """
    transferedList = []
    for token in row:
        try:
            transferedList.append(GLovevectors.stoi[token])
        except KeyError:
            transferedList.append(400000)
    return transferedList

dataset['reviewTexttokenized'] =
dataset['reviewTexttokenized'].apply(Token2EmbedIndex)
dataset['summarytokenized'] =
dataset['summarytokenized'].apply(Token2EmbedIndex)

def paddingfunc(row, length):
    """padding the list to the same length with 0"""
    if len(row) == length:
        pass
    elif len(row) < length:
        for i in range(length - len(row)):
            row.append(0)
    elif len(row) > length:
        row = row[:length]
    return np.array(row)

dataset['reviewTexttokenized'] =
dataset['reviewTexttokenized'].apply(lambda x: paddingfunc(x,length =
config.max_seq_len_review))
```

```
dataset['summarytokenized'] = dataset['summarytokenized'].apply(lambda
x: paddingfunc(x,length = config.max_seq_len_summary))


# In[11]:


# unknown embedding
GLovevectors.vectors =
torch.cat((GLovevectors.vectors,GLovevectors.vectors.mean(axis=0).unsq
ueeze(0)),0)


# In[12]:


ally = dataset[['rating']].apply(lambda x:x-1)
allX = dataset[['reviewTexttokenized',
                'summarytokenized']]

# splitting the test
## =============
X_tr_va, X_test, y_tr_va, y_test = train_test_split(allX,
                                                    ally,
                                                    test_size=1/6,
                                                    random_state=12,
                                                    stratify=ally)

X_train, X_valid, y_train, y_valid = train_test_split(X_tr_va,
                                                      y_tr_va,
                                                      test_size=1/5,
                                                      random_state=12,

stratify=y_tr_va)

#X_tr_va.reset_index(drop = True, inplace = True)
#X_test.reset_index(drop = True, inplace = True)
#y_tr_va.reset_index(drop = True, inplace = True)
#y_test.reset_index(drop = True, inplace = True)

## the rest will be cross validation - use stratified kfold
#SKF = StratifiedKFold(n_splits = 5, random_state = 12, shuffle  =
True)
#DFlist = []
#for train_index, valid_index in SKF.split(X_tr_va, y_tr_va):
#    X_train, X_valid = X_tr_va.iloc[train_index],
X_tr_va.iloc[valid_index]
#    y_train, y_valid = y_tr_va.iloc[train_index],
y_tr_va.iloc[valid_index]
#
#    DFlist.append((X_train, X_valid, y_train, y_valid))


# In[13]:


# DFlist[1][1].head(2)
```

```
# In[14]:


traindata = allX.join(ally)


# In[15]:


ratings = torch.tensor(ally.values, dtype=torch.float)


# In[16]:


def task_importance_weights(label_array):
    uniq = torch.unique(label_array)
    num_examples = label_array.size(0)

    m = torch.zeros(uniq.shape[0])

    for i, t in enumerate(torch.arange(torch.min(uniq),
torch.max(uniq))):
        m_k = torch.max(torch.tensor([label_array[label_array >
t].size(0),
                                      num_examples -
label_array[label_array > t].size(0)]))
        m[i] = torch.sqrt(m_k.float())

    imp = m/torch.max(m)
    return imp

imp = task_importance_weights(ratings)
imp = imp[0:4]


# In[17]:


# dataset loader
class ratingDataset(Dataset):

    def __init__(self, review, summary, rating):

        df = traindata
        self.rating = rating
        self.review = review
        self.summary = summary

    def __getitem__(self, index):
        review = torch.Tensor(self.review.iloc[index]).long()
        summary= torch.Tensor(self.summary.iloc[index]).long()
        label = self.rating.iloc[index]
        levels = [1]*label + [0]*(5 - 1 - label) #encoding the target
        levels = torch.tensor(levels, dtype=torch.float32)

        return review, summary, label, levels
```

```
    def __len__(self):
        return self.rating.shape[0]


# In[18]:


train_dataset = ratingDataset(X_train['reviewTexttokenized'],
                              X_train['summarytokenized'],
                              y_train['rating'])

valid_dataset = ratingDataset(X_valid['reviewTexttokenized'],
                              X_valid['summarytokenized'],
                              y_valid['rating'])

test_dataset = ratingDataset(X_test['reviewTexttokenized'],
                             X_test['summarytokenized'],
                             y_test['rating'])

train_loader = DataLoader(dataset=train_dataset,
                          batch_size=config.batch_size,
                          shuffle=True)

valid_loader = DataLoader(dataset=valid_dataset,
                          batch_size=config.batch_size,
                          shuffle=False)

test_loader = DataLoader(dataset=test_dataset,
                         batch_size=config.batch_size,
                         shuffle=False)


# ## Model

# In[19]:


class BiLSTM_TextCNN(nn.Module):
    def __init__(self, config):
        super(BiLSTM_TextCNN, self).__init__()
        #embedding layer

        self.embedding = nn.Embedding(config.embed_vocab_num,
config.embed_dim)
        self.embedding.weight.data.copy_(GLovevectors.vectors)
        self.embedding.weight.data.requires_grad =
config.embed_trainable

        # Bi-LSTM architecture

        self.BiLSTM = nn.LSTM(input_size = config.embed_dim,
                              hidden_size = config.hidden_size,
                              num_layers = config.lstm_layers,
                              bidirectional=True,
                              # first dimension is batch size
                              batch_first=True,
                              dropout = config.dropout
```

```
                        )
        # output dim (batch, sentense length, hidden size * 2)

        # CNN architecture after Bi LSTM
        self.conv_block_2 = nn.Sequential(
            nn.Conv1d(in_channels = config.hidden_size*2,
                      out_channels = config.kernel_num,
                      kernel_size = config.kernel_size[0]),
            nn.ReLU(),#activate
            nn.MaxPool1d(config.max_seq_len_review -
config.kernel_size[0] + 1) #(n-2+1)*1.
        )

        self.conv_block_3 = nn.Sequential(
            nn.Conv1d(in_channels = config.hidden_size*2,
                      out_channels = config.kernel_num,
                      kernel_size = config.kernel_size[1]),
            nn.ReLU(),#activate
            nn.MaxPool1d(config.max_seq_len_review -
config.kernel_size[1] + 1) #(n-3+1)*1.
        )

        self.conv_block_4 = nn.Sequential(
            nn.Conv1d(in_channels = config.hidden_size*2,
                      out_channels = config.kernel_num,
                      kernel_size = config.kernel_size[2]),
            nn.ReLU(),#activate
            nn.MaxPool1d(config.max_seq_len_review -
config.kernel_size[2] + 1) #(n-4+1)*1.
        )

        # CNN architecture for summary
        #  ================================

        self.conv_block_2_s = nn.Sequential(
            nn.Conv1d(config.embed_dim, config.kernel_num,
config.kernel_size[0]),
            nn.ReLU(),#activate
            nn.MaxPool1d(config.max_seq_len_summary -
config.kernel_size[0] + 1) #(n-2+1)*1.
        )

        self.conv_block_3_s = nn.Sequential(
            nn.Conv1d(config.embed_dim, config.kernel_num,
config.kernel_size[1]),
            nn.ReLU(),#activate
            nn.MaxPool1d(config.max_seq_len_summary -
config.kernel_size[1] + 1) #(n-3+1)*1.
        )

        self.conv_block_4_s = nn.Sequential(
            nn.Conv1d(config.embed_dim, config.kernel_num,
config.kernel_size[2]),
            nn.ReLU(),#activate
            nn.MaxPool1d(config.max_seq_len_summary -
config.kernel_size[2] + 1) #(n-4+1)*1.
        )
        # classify layer ==============================
```

```
        self.dropout = nn.Dropout(config.dropout)

        # 2 cnn + 2 individual input
        self.fc = nn.Linear(config.kernel_num *
len(config.kernel_size)*2, 1) #+2
        self.linear_1_bias = nn.Parameter(torch.zeros(5-1).float())

    def forward(self, review, summary):
        # shape:
        # review = batchsize , max_lengthreview
        # summary = batchsize , max_lengthsummary
        # 2 Uppers = batchsize , 1

        # (bi lstm + cnn) for reviewtokenized
            # Hidden and cell state definion
        #h0 = torch.zeros((2*config.lstm_layers, config.batch_size,
config.hidden_size))
        #c0 = torch.zeros((2*config.lstm_layers, config.batch_size,
config.hidden_size))
                # normal distributed init
        #torch.nn.init.xavier_normal_(h0)
        #torch.nn.init.xavier_normal_(c0)
            # model
        embedded_review = self.embedding(review) # embedded = batch,
length, embedd dim
        packed_output, (hidden, cell) = self.BiLSTM(embedded_review)#
,(h0,c0)
        # packed_output = batch , max length , 2* hidden size
        # packed_output = packed_output.unsqueeze(1)
        packed_output = packed_output.transpose(2,1)
        conv_block_2 = self.conv_block_2(packed_output)
            # input = batch , max length , 2* hidden size
            # conv1dout = batch, kernel num, max length
            #conv_block.shape: (batch_size, kernel_num, 1)
        conv_block_3 = self.conv_block_3(packed_output)
        conv_block_4 = self.conv_block_4(packed_output)

        out_review = torch.cat((conv_block_2.squeeze(2),
                                conv_block_3.squeeze(2),
                                conv_block_4.squeeze(2)), 1)

        # cnn for summarytokenized

        embedded_summary = self.embedding(summary)
        # embedded_summary = embedded_summary.unsqueeze(1)
        embedded_summary = embedded_summary.transpose(2,1)

        conv_block_2_s = self.conv_block_2_s(embedded_summary)
        conv_block_3_s = self.conv_block_3_s(embedded_summary)
        conv_block_4_s = self.conv_block_4_s(embedded_summary)

        out_summary = torch.cat((conv_block_2_s.squeeze(2),
                                 conv_block_3_s.squeeze(2),
                                 conv_block_4_s.squeeze(2)), 1)

        #print(out_review,'=='*10, out_summary,'=='*10,
Upperreview,'=='*10, Uppersummary)
```

```
        #print(Upperreview.shape)
        #out_review.flatten()
        #out_summary.flatten()

        concatfeature = torch.cat((out_review, out_summary),1) #
256*48 256*48 256*1 256*1

        # full connect and softmax
        x = self.dropout(concatfeature)
        logits = self.fc(x)
        logits = logits + self.linear_1_bias
        probas = torch.sigmoid(logits)
        return logits, probas


# In[20]:


def cost_fn(logits, levels, imp):
    val = (-torch.sum((F.logsigmoid(logits)*levels
                      + (F.logsigmoid(logits) -
logits)*(1-levels))*imp,
           dim=1))
    return torch.mean(val)


# In[21]:


model = BiLSTM_TextCNN(config = Config())
optimizer = torch.optim.Adam(model.parameters(),
lr=config.learning_rate)


# In[22]:


from sklearn.metrics import f1_score


# In[23]:


def compute_F1_score(model, data_loader, device):
    f1score, num_examples = 0, 0
    targetlist = []
    predictedlist = []
    for i, (review,summary, targets, levels) in
enumerate(data_loader):

        logits, probas = model(review,summary)
        predict_levels = probas > 0.5
        predicted_labels = torch.sum(predict_levels, dim=1)
        targetlist.extend(targets.tolist())
        predictedlist.extend(predicted_labels.tolist())

    f1Score = f1_score(predictedlist, targetlist, average =
'weighted')
```

```
    return f1Score


# In[24]:


start_time = time.time()
best_f1, best_epoch = 0, -1
for epoch in range(config.epoch):

    model.train()
    for batch_idx, (review,summary, targets, levels) in
enumerate(train_loader):

        # FORWARD AND BACK PROP
        logits, probas = model(review,summary)
        cost = cost_fn(logits, levels, imp)
        optimizer.zero_grad()

        cost.backward()

        # UPDATE MODEL PARAMETERS
        optimizer.step()

        # LOGGING
        if not batch_idx % 50:
            print('Epoch: %03d/%03d | Batch %04d/%04d | Cost: %.4f'
                   % (epoch+1, config.epoch , batch_idx,
                       len(train_dataset)//config.batch_size, cost))

    model.eval()
    with torch.set_grad_enabled(False):
        valid_f1 = compute_F1_score(model, valid_loader,'cpu')
        #valid_mae, valid_mse = compute_mae_and_mse(model,
valid_loader, 'cpu')

    if valid_f1 > best_f1:
        best_f1, best_epoch = valid_f1, epoch
        ########## SAVE MODEL #############
        torch.save(model.state_dict(), os.path.join(r"D:/program
files/jupyter notebook/usyd\6850/Final Version/", 'best_model.pt'))


    s = 'f1: | Current Valid: %.4f Ep. %d | Best Valid : %.4f Ep. %d'
% (
        valid_f1, epoch, best_f1, best_epoch)
    print(s)

    s = 'Time elapsed: %.2f min' % ((time.time() - start_time)/60)
    print(s)

model.eval()


# In[25]:


with torch.set_grad_enabled(False):  # save memory during inference
```

```
    train_f1 = compute_F1_score(model, train_loader,'cpu')
    valid_f1 = compute_F1_score(model, valid_loader,'cpu')
    test_f1 = compute_F1_score(model, test_loader,'cpu')

    s = 'f1 score: | Train: %.4f | Valid: %.4f | Test: %.4f' % (
        train_f1,
        valid_f1,
        test_f1, )
    print(s)

s = 'Total Training Time: %.2f min' % ((time.time() - start_time)/60)
print(s)


# In[26]:


# evaluate best model
model.load_state_dict(torch.load(r"D:/program files/jupyter
notebook/usyd/6850/Final Version/best_model.pt"))####
model.eval()

with torch.set_grad_enabled(False):
    train_f1 = compute_F1_score(model, train_loader,
                                               device='cpu')
    valid_f1 = compute_F1_score(model, valid_loader,
                                               device='cpu')
    test_f1 = compute_F1_score(model, test_loader,
                                             device='cpu')

    s = 'f1: | Best Train: %.4f | Best Valid: %.4f | Best Test: %.4f'
% (
        train_f1,
        valid_f1,
        test_f1)
    print(s)


# In[27]:


# save predictions
all_pred = []
all_probas = []
with torch.set_grad_enabled(False):
    for batch_idx, (review,summary, targets, levels) in
enumerate(test_loader):


        logits, probas = model(review,summary)
        all_probas.append(probas)
        predict_levels = probas > 0.5
        predicted_labels = torch.sum(predict_levels, dim=1)
        lst = [str(int(i)) for i in predicted_labels]
        all_pred.extend(lst)
```

```
torch.save(torch.cat(all_probas).to(torch.device('cpu')),r"D:/program
files/jupyter notebook/usyd/6850/Final
Version/test_allprobas.tensor")####


# In[28]:


test_pred = pd.DataFrame(data = all_pred, columns = ['rating'])
test_pred['rating'] = test_pred['rating'].apply(lambda x: int(x))
print('accuracy:')
print((test_pred['rating'].values ==
y_test['rating'].values).sum()/1500)


# In[34]:


f1_score(test_pred['rating'].values, y_test['rating'].values, average
= 'macro')


# In[29]:


#submission
subdataset = pd.read_csv('preprocessedtest_deep.csv',
                         index_col = 0,
                         converters = {'reviewTexttokenized': eval,
                                       'summarytokenized': eval})

subdataset['reviewTexttokenized'] =
subdataset['reviewTexttokenized'].apply(Token2EmbedIndex)
subdataset['summarytokenized'] =
subdataset['summarytokenized'].apply(Token2EmbedIndex)

subdataset['reviewTexttokenized'] =
subdataset['reviewTexttokenized'].apply(lambda x: paddingfunc(x,length
= config.max_seq_len_review))
subdataset['summarytokenized'] =
subdataset['summarytokenized'].apply(lambda x: paddingfunc(x,length =
config.max_seq_len_summary))


# In[30]:


subdf = subdataset[['reviewTexttokenized','summarytokenized']]


# In[31]:


subdf


# In[32]:
```

```
sub_prob = []
subpredict = []
with torch.set_grad_enabled(False):
    for row in subdf.iterrows():
        logits, probas =
model(torch.unsqueeze(torch.from_numpy(row[1]['reviewTexttokenized']),
0),

torch.unsqueeze(torch.from_numpy(row[1]['summarytokenized']),0))
        sub_prob.append(probas)
        predict_levels = probas > 0.5
        predicted_labels = torch.sum(predict_levels, dim=1)
        lst = [str(int(i)+1) for i in predicted_labels]
        subpredict.extend(lst)


# In[33]:


outputdf = pd.DataFrame(data=subpredict)
outputdf.index.names = (['id'])
outputdf.to_csv('submission_glove_Bilstm_cnn_ordinal_deep.csv', header
= ['prediction'])
```

TASK C

```
import os
os.chdir('C:/Users/lmk/Desktop/sydney/sem3/6850/project')
import pandas as pd
import math
import gc
import time
import tqdm
import random
import torch
print(torch.__version__)
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
#device = torch.device("cpu")
print(device)
from torch import nn
from torch import utils
from torch.nn.utils.rnn import pad_sequence

from torch.nn import Parameter
import torch.nn.functional as F
from torch.optim import Adam, SGD, AdamW
from torch.utils.data import DataLoader, Dataset


from torchtext.data.utils import get_tokenizer
from torchtext.vocab import vocab
from collections import Counter
import numpy as np
```

```
import scipy as sp
from transformers import BertTokenizer, BertModel, BertForMaskedLM
from sklearn.model_selection import StratifiedKFold, GroupKFold, KFold
,train_test_split

import tokenizers
import transformers
print(f"tokenizers.__version__: {tokenizers.__version__}")
print(f"transformers.__version__: {transformers.__version__}")
from transformers import AutoTokenizer, AutoModel, AutoConfig
from transformers import get_linear_schedule_with_warmup,
get_cosine_schedule_with_warmup

INPUT_DIR = 'C:/Users/lmk/Desktop/sydney/sem3/6850/project/'
OUTPUT_DIR = 'C:/Users/lmk/Desktop/sydney/sem3/6850/project/output/'

from sklearn.metrics import f1_score
# ====================================================
# CFG
# ====================================================
class CFG:
    num_workers=0 #填4 多线程报错
    path="C:/Users/lmk/Desktop/sydney/sem3/6850/project/\
        input/pppm-deberta-v3-large-baseline-w-w-b-train/"
    config_path=path+'config.pth'
    #model="microsoft/deberta-v3-large"
    model = "distilbert-base-uncased"
    #model = 'bert-base-uncased'
    batch_size=2
    fc_dropout=0.2
    target_size=1

    max_len=2693 #之后定义

    seed=42
    n_fold=4
    trn_fold=[0, 1, 2, 3]
    encoder_lr=2e-7
    decoder_lr=2e-7
    min_lr=1e-8
    weight_decay=0.01
    eps=1e-6            #让adam分母不为0
    betas=(0.9, 0.999)   #adam 保留前一，二个时刻learning rate 的比例
    epochs=4
    scheduler='cosine' # ['linear', 'cosine']  学习率调度器
    num_warmup_steps=0   #耐心系数 lr先慢慢增加，超过warmup_steps时，lr再慢慢
减小。
    num_cycles=0.5    #学习率第一段线性增加   之后像余弦函数一样先减后增循环的次
数 0.5代表只减
    tokenizer = None #之后添加
    apex=False      #数据精度自动匹配 缩短训练时间，降低存储需求， 用mse的时候会
报错 在之后的版本可能修复
                         #因而能支持更多的 batch size、更大模型和尺寸更大的输入进行训
练
    gradient_accumulation_steps = 8 #通过累计梯度来解决本地显存不足问题。在
loss函数的时候要用
    max_grad_norm = 1  #对parameters里的所有参数的梯度进行规范化
```

```
                #梯度裁剪解决的是梯度消失或爆炸的问题，即设定阈值，如果梯度超过阈值，那么就截断，将梯度变为阈值
    print_freq = 10
    batch_scheduler=True
# ====================================================
# Utils  分数是相关系数
# ====================================================
def get_score(y_true, y_pred):
    #score = sp.stats.pearsonr(y_true, y_pred)[0]
    score =f1_score(y_pred, y_true, average = 'macro')
    return score

def get_logger(filename=OUTPUT_DIR+'train'):
    from logging import getLogger, INFO, StreamHandler, FileHandler, Formatter
    logger = getLogger(__name__)
    logger.setLevel(INFO)
    handler1 = StreamHandler()
    handler1.setFormatter(Formatter("%(message)s"))
    handler2 = FileHandler(filename=f"{filename}.log")
    handler2.setFormatter(Formatter("%(message)s"))
    logger.addHandler(handler1)
    logger.addHandler(handler2)
    return logger

LOGGER = get_logger()

def seed_everything(seed=12):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True

seed_everything(seed=12)

# ===========================================================================
# OOF  out of frame?
# ===========================================================================
#oof_df = pd.read_pickle(CFG.path+'oof_df.pkl')
#labels = oof_df['score'].values
#preds = oof_df['pred'].values
#score = get_score(labels, preds)
#LOGGER.info(f'CV Score: {score:<.4f}')


# ====================================================
# Data Loading
# ====================================================
dataset = pd.read_csv('preprocessedtrain.csv',
                      index_col = 0,
                      converters = {'reviewTexttokenized': eval,
                                    'summarytokenized': eval}
```

```
                     )

dataset = dataset[['rating','reviewText','summary']]
# splitting the test

train,test = train_test_split(dataset,
                            test_size=1/6,
                            random_state=12,
                            stratify=dataset['rating'])

train.reset_index(drop = True, inplace=True)
test.reset_index(drop = True, inplace=True)
# ====================================================
# CV split
# ====================================================
SKF = StratifiedKFold(n_splits = 5, random_state = 12, shuffle  =
True)


DFlist = []
for train_index, valid_index in SKF.split(train,train['rating']):
    train1, valid1 = train.iloc[train_index], train.iloc[valid_index]
    DFlist.append((train1, valid1))

#train['rating'].hist()

# ====================================================
# tokenizer
# ====================================================
tokenizer = AutoTokenizer.from_pretrained(CFG.model,
do_lower_case=True)
tokenizer.save_pretrained(OUTPUT_DIR+'tokenizer/')
CFG.tokenizer = tokenizer

# ====================================================
# Define max_len
# ====================================================

review_lengths = []
#tk0 = tqdm(train['text'].unique(), total=len(train['text'].unique()))
#训练集每句话的长度
for text in train['reviewText'].unique():
    length = len(tokenizer(text,
add_special_tokens=False)['input_ids'])
    review_lengths.append(length)

summary_lengths = []

for text in train['summary'].unique():
    length = len(tokenizer(text,
add_special_tokens=False)['input_ids'])
    summary_lengths.append(length)


max(review_lengths)  #2693

max(summary_lengths) #33
```

```
#
======================================================================
=======
# for text_col in ['anchor', 'target']:
#     lengths = []
#     tk0 = tqdm(train[text_col].fillna("").values, total=len(train))
#     for text in tk0:
#         length = len(tokenizer(text,
add_special_tokens=False)['input_ids'])
#         lengths.append(length)
#     lengths_dict[text_col] = lengths
#
======================================================================
=======
#
======================================================================
=======
#
# CFG.max_len = max(lengths_dict['anchor']) +
max(lengths_dict['target'])\
#                  + max(lengths_dict['context_text']) + 4 # CLS + SEP
+ SEP + SEP
#
======================================================================
=======
#CFG.max_len = max(lengths)

if max(review_lengths)>512: #用蒸馏模型 能处理最大为512
  CFG.max_len = 512
else:
  CFG.max_len = max(review_lengths)

LOGGER.info(f"max_len: {CFG.max_len}")

# ====================================================
# Dataset
# ====================================================
def prepare_input(cfg, review,summary):
    reviews = cfg.tokenizer(review,
                           add_special_tokens=True,
                           max_length=cfg.max_len,
                           padding="max_length", #补全
                           return_offsets_mapping=False,
                           truncation=True #截断  'only_first'：这个只针
对第一个序列。'only_second'：只针对第二个序列。
                           )
    summarys = cfg.tokenizer(summary,
                           add_special_tokens=True,
                           max_length=cfg.max_len,
                           padding="max_length", #补全
                           return_offsets_mapping=False,
                           truncation=True #截断  'only_first'：这个只针
对第一个序列。'only_second'：只针对第二个序列。
                           )
    #这样写不能改 因为tokenizer出来是一种特殊形式  review【0】是输入的词向量【1
】是mask
    for k, v in reviews.items():
        reviews[k] = torch.tensor(v, dtype=torch.long)
```

```
    for k, v in summarys.items():
        summarys[k] = torch.tensor(v, dtype=torch.long)
    return reviews,summarys


class TrainDataset(Dataset):
    def __init__(self, cfg, df):
        self.cfg = cfg
        self.reviews = df['reviewText'].values
        self.summarys = df['summary'].values

        self.labels = df['rating'].values

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

    def __getitem__(self, item):
        reviews,summarys = prepare_input(self.cfg,
self.reviews[item],self.summarys[item])
        label = torch.tensor(self.labels[item], dtype=torch.float)
        return reviews,summarys, label


# ====================================================
# Model
# ====================================================
class CustomModel(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.cfg = cfg
        self.config = AutoConfig.from_pretrained(cfg.model,
output_hidden_states=True)
        self.model = AutoModel.from_config(self.config)
        self.fc_dropout = nn.Dropout(cfg.fc_dropout)

        self.fc = nn.Linear(self.config.hidden_size * 2,
self.cfg.target_size) #有两个bert *2
        self._init_weights(self.fc)

        self.attention = nn.Sequential(
            nn.Linear(self.config.hidden_size, 512),
            nn.Tanh(),
            nn.Linear(512, 1),
            nn.Softmax(dim=1)
        )
        self._init_weights(self.attention)

    def _init_weights(self, module):
        if isinstance(module, nn.Linear):
            module.weight.data.normal_(mean=0.0,
std=self.config.initializer_range)
            if module.bias is not None:
                module.bias.data.zero_()
        elif isinstance(module, nn.Embedding):
            module.weight.data.normal_(mean=0.0,
std=self.config.initializer_range)
            if module.padding_idx is not None:
```

```
                module.weight.data[module.padding_idx].zero_()
        elif isinstance(module, nn.LayerNorm):
            module.bias.data.zero_()
            module.weight.data.fill_(1.0)

    def feature(self, inputs):
        outputs = self.model(**inputs)   #model在初始化时就已经是预训练的bert模型了
        last_hidden_states = outputs[0]
        # feature = torch.mean(last_hidden_states, 1)
        weights = self.attention(last_hidden_states)
        feature = torch.sum(weights * last_hidden_states, dim=1)
        return feature

    def forward(self, reviews,summarys):
        feature1 = self.feature(reviews)   #feature 就是预训练模型给出的隐含特征

        feature2 = self.feature(summarys)

        feature = torch.cat((feature1,feature2),1)


        output = self.fc(self.fc_dropout(feature))
        return output

class AverageMeter(object):
    """Computes and stores the average and current value"""
    def __init__(self):
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count


def asMinutes(s):
    m = math.floor(s / 60)
    s -= m * 60
    return '%dm %ds' % (m, s)


def timeSince(since, percent):
    now = time.time()
    s = now - since
    es = s / (percent)
    rs = es - s
    return '%s (remain %s)' % (asMinutes(s), asMinutes(rs))
```

```
def train_fn(fold, train_loader, model, criterion, optimizer, epoch,
scheduler, device):
    model.train()
    scaler = torch.cuda.amp.GradScaler(enabled=CFG.apex)
    losses = AverageMeter()
    start = end = time.time()
    global_step = 0

    for step, (reviews,summarys, labels) in enumerate(train_loader):


        for k, v in reviews.items():
            reviews[k] = v.to(device)
        for k, v in summarys.items():
            summarys[k] = v.to(device)

        labels = labels.to(device)
        batch_size = labels.size(0)



        with torch.cuda.amp.autocast(enabled=CFG.apex):   #这一步导致mse
error时调用c++内核报错
            y_preds = model(reviews,summarys)

        loss = criterion(y_preds.view(-1, 1).to(device),
labels.view(-1, 1))

        if CFG.gradient_accumulation_steps > 1:
            loss = loss / CFG.gradient_accumulation_steps
        losses.update(loss.item(), batch_size)


        scaler.scale(loss).backward()


        grad_norm = torch.nn.utils.clip_grad_norm_(model.parameters(),
CFG.max_grad_norm)
        if (step + 1) % CFG.gradient_accumulation_steps == 0:
            scaler.step(optimizer)
            scaler.update()
            optimizer.zero_grad()
            global_step += 1
            if CFG.batch_scheduler:
                scheduler.step()
        end = time.time()
        if step % CFG.print_freq == 0 or step ==
(len(train_loader)-1):
            print('Epoch: [{0}][{1}/{2}] '
                  'Elapsed {remain:s} '
                  'Loss: {loss.val:.4f}({loss.avg:.4f}) '
                  'Grad: {grad_norm:.4f}  '
                  'LR: {lr:.8f}  '
                  .format(epoch+1, step, len(train_loader),
                          remain=timeSince(start,
float(step+1)/len(train_loader)),
                          loss=losses,
```

```
                              grad_norm=grad_norm,
                              lr=scheduler.get_lr()[0]))

    return losses.avg


def valid_fn(valid_loader, model, criterion, device):
    losses = AverageMeter()
    model.eval()
    preds = []
    start = end = time.time()
    for step, (reviews,summarys, labels) in enumerate(valid_loader):

        for k, v in reviews.items():
            reviews[k] = v.to(device)
        for k, v in summarys.items():
            summarys[k] = v.to(device)
        batch_size = labels.size(0)

        labels = labels.to(device)

        with torch.no_grad():
            y_preds = model(reviews,summarys)
        loss = criterion(y_preds.view(-1, 1).to(device),
labels.view(-1, 1))  #顺序不能变 ！！！！！
        if CFG.gradient_accumulation_steps > 1:
            loss = loss / CFG.gradient_accumulation_steps
        losses.update(loss.item(), batch_size)
        preds.append(y_preds.sigmoid().to('cpu').numpy())
        end = time.time()
        if step % CFG.print_freq == 0 or step ==
(len(valid_loader)-1):
            print('EVAL: [{0}/{1}] '
                  'Elapsed {remain:s} '
                  'Loss: {loss.val:.4f}({loss.avg:.4f}) '
                  .format(step, len(valid_loader),
                          loss=losses,
                          remain=timeSince(start,
float(step+1)/len(valid_loader))))
    predictions = np.concatenate(preds)
    predictions = np.concatenate(predictions)
    return losses.avg, predictions


def inference_fn(test_loader, model, device):
    preds = []
    model.eval()
    model.to(device)
    tk0 = tqdm(test_loader, total=len(test_loader))
    for reviews,summarys, labels in tk0:
        for k, v in reviews.items():
            reviews[k] = v.to(device)
        for k, v in summarys.items():
            summarys[k] = v.to(device)

        with torch.no_grad():
            y_preds = model(reviews,summarys)
        preds.append(y_preds.sigmoid().to('cpu').numpy())
```

```
    predictions = np.concatenate(preds)
    return predictions

# ====================================================
# train loop
# ====================================================
def train_loop(DFlist,n,treshold):

    LOGGER.info(f"========== fold: {fold} training ==========")

    # ====================================================
    # loader
    # ====================================================
    train_folds = DFlist[n][0]
    #train_folds['text'] = train_folds['text'].astype(str)

    valid_folds = DFlist[n][1]
    #valid_folds['text'] = valid_folds['text'].astype(str)
    valid_labels = valid_folds['rating'].values #validation 函数调用了
所以要提出来用

    train_dataset = TrainDataset(CFG, train_folds)
    valid_dataset = TrainDataset(CFG, valid_folds)

    train_loader = DataLoader(train_dataset,
                              #collate_fn = lambda x: collate_batch(x,
CFG), #传参
                              batch_size=CFG.batch_size,
                              shuffle=True,
                              num_workers=CFG.num_workers,
pin_memory=True, drop_last=True)
    valid_loader = DataLoader(valid_dataset,
                              #collate_fn = lambda x: collate_batch(x,
CFG), #传参
                              batch_size=CFG.batch_size,
                              shuffle=False,
                              num_workers=CFG.num_workers,
pin_memory=True, drop_last=False)

    # ====================================================
    # model & optimizer
    # ====================================================
    model = CustomModel(CFG)
    #torch.save(model.config, OUTPUT_DIR+'config.pth')
    model.to(device)

    def get_optimizer_params(model, encoder_lr, decoder_lr,
weight_decay=0.0):
        param_optimizer = list(model.named_parameters())
        no_decay = ["bias", "LayerNorm.bias", "LayerNorm.weight"]
        optimizer_parameters = [
            {'params': [p for n, p in model.model.named_parameters()
if not any(nd in n for nd in no_decay)],
             'lr': encoder_lr, 'weight_decay': weight_decay},
            {'params': [p for n, p in model.model.named_parameters()
if any(nd in n for nd in no_decay)],
             'lr': encoder_lr, 'weight_decay': 0.0},
```

```
            {'params': [p for n, p in model.named_parameters() if
"model" not in n],
             'lr': decoder_lr, 'weight_decay': 0.0}
        ]
        return optimizer_parameters

    optimizer_parameters = get_optimizer_params(model,

encoder_lr=CFG.encoder_lr,

decoder_lr=CFG.decoder_lr,

weight_decay=CFG.weight_decay)
    optimizer = AdamW(optimizer_parameters, lr=CFG.encoder_lr,
eps=CFG.eps, betas=CFG.betas)

    # ====================================================
    # scheduler
    # ====================================================
    def get_scheduler(cfg, optimizer, num_train_steps):
        if cfg.scheduler == 'linear':
            scheduler = get_linear_schedule_with_warmup(
                optimizer, num_warmup_steps=cfg.num_warmup_steps,
num_training_steps=num_train_steps
            )
        elif cfg.scheduler == 'cosine':
            scheduler = get_cosine_schedule_with_warmup(
                optimizer, num_warmup_steps=cfg.num_warmup_steps,
num_training_steps=num_train_steps, num_cycles=cfg.num_cycles
            )
        return scheduler

    num_train_steps = int(len(train_folds) / CFG.batch_size *
CFG.epochs)
    scheduler = get_scheduler(CFG, optimizer, num_train_steps)

    # ====================================================
    # loop
    # ====================================================
    #criterion =
nn.BCEWithLogitsLoss(reduction="mean")#Sigmoid+BCELoss 设为"sum"表示对样
本进行求损失和；
                                                              #设为"mean"表示对样
本进行求损失的平均值；
                                                              #而设为"none"表示对
样本逐个求损失，输出与输入的shape一样。
    criterion = nn.MSELoss(reduction="mean") #平均值
    best_score = 0.

    for epoch in range(CFG.epochs):

        start_time = time.time()

        # train
        avg_loss = train_fn(fold, train_loader, model, criterion,
optimizer, epoch, scheduler, device)

        # eval
```

```
        avg_val_loss, predictions = valid_fn(valid_loader, model, 
criterion, device)

        # scoring

        treshold[0].append(np.quantile(predictions,0.266667))
        treshold[1].append(np.quantile(predictions,0.511111))
        treshold[2].append(np.quantile(predictions,0.7))
        treshold[3].append(np.quantile(predictions,0.866667))

        tre1 = np.mean(treshold[0][-30:])
        tre2 = np.mean(treshold[1][-30:])
        tre3 = np.mean(treshold[2][-30:])
        tre4 = np.mean(treshold[3][-30:])

        classified_y = 
1+(predictions>tre1).astype(int)+(predictions>tre2).astype(int)+(predi
ctions>tre3).astype(int)+(predictions>tre4).astype(int)

        score = get_score(valid_labels, classified_y)

        elapsed = time.time() - start_time

        LOGGER.info(f'Epoch {epoch+1} - avg_train_loss: {avg_loss:.4f}  
avg_val_loss: {avg_val_loss:.4f}  time: {elapsed:.0f}s')
        LOGGER.info(f'Epoch {epoch+1} - Score: {score:.4f}')

        if best_score < score:
            best_score = score
            LOGGER.info(f'Epoch {epoch+1} - Save Best Score: 
{best_score:.4f} Model')

            torch.save({'model': model.state_dict(),
                        'predictions': predictions},
                        OUTPUT_DIR+f"{CFG.model.replace('/', 
'-')}_fold{fold}_best.pth")




    predictions = torch.load(OUTPUT_DIR+f"{CFG.model.replace('/', 
'-')}_fold{fold}_best.pth", 

map_location=torch.device('cpu'))['predictions']




    valid_folds['pred'] = predictions

    torch.cuda.empty_cache()
    gc.collect()

    return valid_folds,treshold

def get_result(oof_df):
    labels = oof_df['rating'].values
    preds = oof_df['pred'].values
```

```
    score = get_score(labels, preds)
    LOGGER.info(f'Score: {score:<.4f}')



treshold=[[],[],[],[]]

oof_df = pd.DataFrame()
for fold in range(1):  #不使用cross-valid
    if fold in CFG.trn_fold:
        _oof_df,treshold = train_loop(DFlist,fold,treshold)# 0

        oof_df = pd.concat([oof_df, _oof_df])
        LOGGER.info(f"========== fold: {fold} result ==========")
        get_result(_oof_df)
    oof_df = oof_df.reset_index(drop=True)
    LOGGER.info(f"========== CV ==========")
    get_result(oof_df)
    oof_df.to_pickle(OUTPUT_DIR+'oof_df.pkl')
```