SDML-HW1 Report

2019 AI Cup Abstract Classification

Team: AndyIsMyBoss
Member: Chieh-Fang Teng D06943020, Yi-Ta Chen D06943017

For more details, please refer to our presentation slides

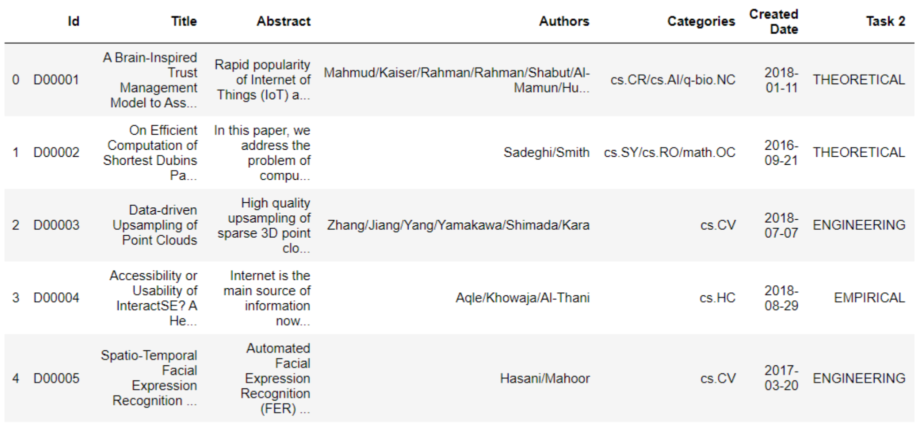
2019/10/20

Contents

1 Problem Formulation

In this task, we are given a set of papers from arXiv. Given title, abstract, author, category, and date from paper, we are asked to predict the paper's type in four classes: theoretical paper, engineering paper, empirical paper, or others.

In order to show the data from the task, we select the first 5 papers and print out the content of each paper. From Fig. 1, we can observe that the title is a sentence and the abstract is constructed with sentences separated by '$$$'. Moreover, authors and categories are also separated by '/'. Therefore, we have to preprocess the data first to obtain the right format.



	Id	Title	Abstract	Authors	Categories	Created Date	Task 2
0	D00001	A Brain-Inspired Trust Management Model to Ass...	Rapid popularity of Internet of Things (IoT) a...	Mahmud/Kaiser/Rahman/Rahman/Shabut/Al-Mamun/Hu...	cs.CR/cs.AI/q-bio.NC	2018-01-11	THEORETICAL
1	D00002	On Efficient Computation of Shortest Dubins Pa...	In this paper, we address the problem of compu...	Sadeghi/Smith	cs.SY/cs.RO/math.OC	2016-09-21	THEORETICAL
2	D00003	Data-driven Upsampling of Point Clouds	High quality upsampling of sparse 3D point clo...	Zhang/Jiang/Yang/Yamakawa/Shimada/Kara	cs.CV	2018-07-07	ENGINEERING
3	D00004	Accessibility or Usability of InteractSE? A He...	Internet is the main source of information now...	Aqle/Khowaja/Al-Thani	cs.HC	2018-08-29	EMPIRICAL
4	D00005	Spatio-Temporal Facial Expression Recognition ...	Automated Facial Expression Recognition (FER) ...	Hasani/Mahoor	cs.CV	2017-03-20	ENGINEERING

Figure 1: An overview of the dataset.

In the following section, we will introduce the processing flow, graph-based embedding, text-based embedding, and joint graph and text embedding with some utilized techniques to improve the performance.

2 Processing Flow

As shown in Fig. 2, the overall processing flow can be seperated into few steps. First, we have to read the dataset of the papers which is already organized in a csv format. After reading the dataset, we have to load it into a dataframe structure using pandas dataframe. Although each field of the dataframe has different processing flow, we can categorize it into graph-based embedding and text-based embedding, which will be discussed in section 3 and section 4, respectively. Finally, the constructed embeddings can be used to train the final classifier.

When processing the abstract, the sentences needed to be separated by '$$$' and saved into a list. Because each paper can be categorized into one of four classes, the label of each paper is a four-dimension vector where each of the field is one if it belongs to the corresponding class, otherwise, it will be zero.

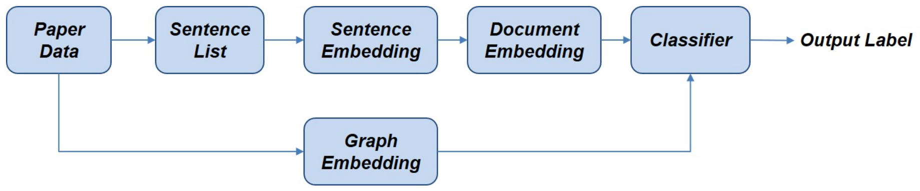


Figure 2: An overview of processing flow.

After all the field has been preprocessed, we can split the dataset into training and validation set to observe the performance of the training process. Finally, we pick a well-trained model based on validation results and apply it on the preprocessed testing data and get the result.

3 Graph-based Embedding

According to the data we have, we tried to utilize citation, author list, and category to construct graph and extract embedding from the network. In this section, we will introduce how we process each kind of data and which graph-based method we used as below.

3.1 Citation

To collect all papers we have, we firstly normalize all papers from paper_title _abstract.csv, task2_trainset.csv, and task2_public_testset.csv for matching. Now, we totally have about 744K papers with 1372K connections. After the construction of the citation-based graph, we utilize Node2Vec to generate 64 dimensional embedding. Then, the generated embedding can be concatenated with output of GRU for further classification by fully connected layer as shown in Fig. 3. The performance after concatenating with citation graph embedding will be analyzed and compared in the end of this section.

3.2 Author

The graph of author is hard to construct because the dataset only provide the author list with each author's last name as shown in Fig. 4. As a result, there must result in a problem that some people has the same last name, but actually not the same people. To solve this problem, we adopt a naive approach that we only pick the row has exactly same author list. Therefore, we can make sure that the choosed papers are related. Based on this criteria, we have totally 1129 groups having the same author list with 2766 papers in them and 3203 connections. Then, we also utilize Node2Vec to generate 64 dimensional embedding and concatenate with GRU output as the method of citation graph embedding.

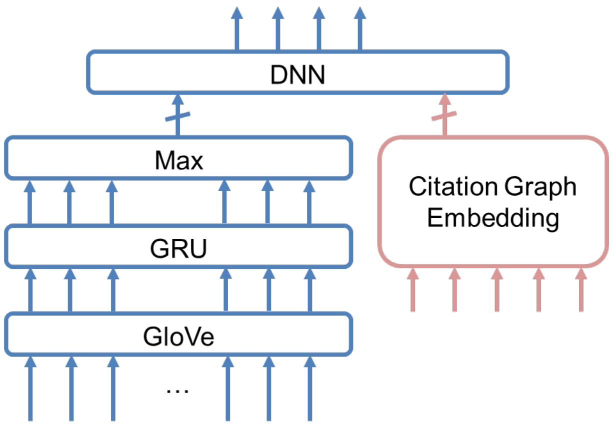
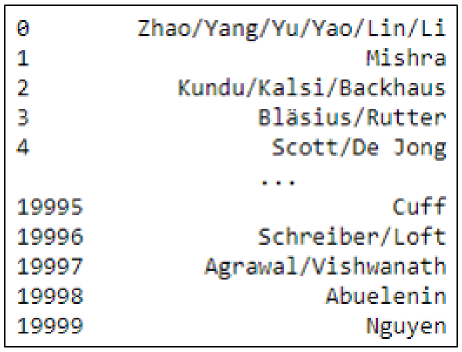


Figure 3: Concatenate citation graph embedding with output of GRU.



```
0        Zhao/Yang/Yu/Yao/Lin/Li
1                         Mishra
2          Kundu/Kalsi/Backhaus
3                 Bläsius/Rutter
4                  Scott/De Jong
                     ...
19995                       Cuff
19996             Schreiber/Loft
19997         Agrawal/Vishwanath
19998                  Abuelenin
19999                     Nguyen
```

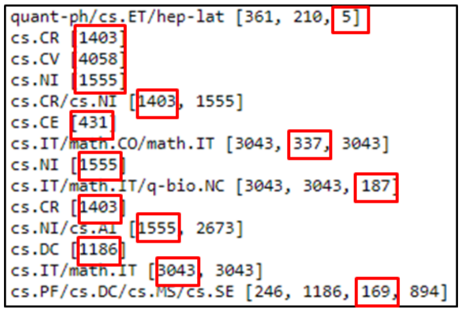
Figure 4: The provided author list in dataset.

3.3 Category

Another graph we can construct is based on category, which is reasonable that two papers have the same category may have some relation between them. After analysis, there are overall 156 different categories in the training and testing dataset. We have tried three different methods to construct the category graph. Firstly, the most intuitive approach is for each category, we connect all papers have this category. However, based on this approach, the number of connections will grow in $O(n^2)$ and results in about 47M connections with 302 hours for the computing of transition probabilities in Node2Vec. After analysis, we find out that there are some categories having more than 2000 papers, which means the connection of these categories may be meaningless for the training of Node2Vec.

Secondly, to solve the aforementioned problem, we come up with another idea by avoiding the connection of categories having more than 1500 papers. Therefore, we can avoid the meaningless connections in the graph and reduce the connections to about 9M with 15 hours for the computing of transition probabilities. However, we find out another serious problem that the second approach causes some paper to have no connections with other papers, thus these papers cannot be trained.

Therefore, we propose the third method. The main idea is that the rarer the category, the closer links between the papers having this category. Thus, for each paper, which is the row in Fig. 5, we choose the rarest category for graph construction. Furthermore, if the rarest category for each paper has more than 200 papers, we randomly sample 200 papers from the category as connections. As a result, we have about 5M connections. After the construction of the category graph, we also utilize Node2Vec to generate 64 dimensional embedding. This method not only can reduce the computing time of transition probabilities to only about 2 hours, but also can make sure that each paper has connections with other papers.



```
quant-ph/cs.ET/hep-lat [361, 210, 5]
cs.CR [1403]
cs.CV [4058]
cs.NI [1555]
cs.CR/cs.NI [1403, 1555]
cs.CE [431]
cs.IT/math.CO/math.IT [3043, 337, 3043]
cs.NI [1555]
cs.IT/math.IT/q-bio.NC [3043, 3043, 187]
cs.CR [1403]
cs.NI/cs.AI [1555, 2673]
cs.DC [1186]
cs.IT/math.IT [3043, 3043]
cs.PF/cs.DC/cs.MS/cs.SE [246, 1186, 169, 894]
```

Figure 5: The illustration of choosed category for each paper.

Finally, we have three different kinds of embedding, which extracted from citation, author, and category, respectively. The F1 score of different kinds of embedding approaches is summarized in Table. 1 as below:

Table 1: The F1 score of different embedding

Baseline: GRU	GRU w/ Citation Embedding	GRU w/ Author Embedding	GRU w/ Category Embedding
0.6771	0.6830	0.6791	0.6794

4 Text-based Embedding

In this section, we will introduce the text-based embedding method to solve the classification problem. In the text-based embedding method, there are two steps: First, you have to encode the sentence. Then the encoded sentences in an abstract will be passed to a classifier to predict the final label. In this work, we have implemented three types of text-based embedding method, GloVe with GRU, Doc2Vec, and BERT.

4.1 GloVe + GRU

In this section, we will introduce the method of encoding using GloVe and trained the GRU classifier. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. We selected the pretrained model to encode every word in a sentence. There are several pretrained models of GloVe. We selected the model that is trained on wikipedia 2014 and gigaword 5 which has total six billions of tokens.

After encoding every words in each sentence, we have to pad the abstract into same size of sentences and same size of words in each sentence. After padding, we construct a gated recurrent unit (GRU) layer with neuron size equal to 512 and input each word enbedding into the GRU. The output of the GRU has the size of $\#\_of\_sentence \times \#\_of\_words \times embedding\_dimension$.We first choose the largest value among word dimension and then choose the largest value among the sentence dimension. Finally, we add up the two dimension value of the GRU output and generate a 512-dimension vector.

The final step is to load the 512-dimension vector into a linear layer to calculate the probability of the four output labels. We use binary cross entropy as our loss function and adam with learning rate equals to $2 \cdot 10^{-4}$. The loss during the training process is shown in Fig. 6 as follows:

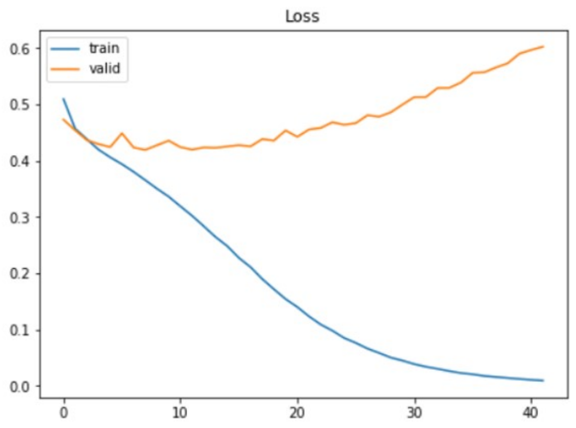


Figure 6: The loss during training of Glove + GRU model.

From Fig. 6, we can find out that after about 5 epochs, the validation loss does not decrease. This shows that the model has serious overfitting problem. We will deal with this problem in Section 6.1. The final best f1 score of this model on validation dataset is *0.6771*.

4.2 Doc2Vec

Another method for text-based embedding is Doc2Vec, which heavily based on Word2Vec by adding paragraph id vector to Word2Vec model. In this section, we try four different ways for the input data of Doc2Vec model. Firstly, the most intuitive approach is to collect all papers from paper_title_abstract.csv, task2_trainset.csv, and task2_public_testset.csv. Thus, we totally have 685163 papers. After sending the abstract from these papers to the Doc2Vec model, we can obtain embedding for these papers.

Secondly, we can imagine that the title for each paper is also informative and can also be seen as another sentence in the abstract. Therefore, we treat the title as the first sentence in the abstract and concatenate with abstract.

For the third and fourth methods, we only focus on the train/test dataset without the paper in paper_title_abstract.csv, which may have better performance in this task. Thus, the number of paper is reduced to 27000. The F1 score for these four different approaches is listed in Table. 2 as below:

Table 2: The F1 score of different input data for Doc2Vec model

Dimension	All Abstract	All Title + Abstract	Train/Test Abstract	Train/Test Title + Abstract
32	0.6100	0.6151	0.6185	0.6283
64	0.6202	0.6034	0.6166	0.6259

According to Table. 2, we can demonstrate that title is informative for further improvement of F1 score. Besides, for the training of embedding, we only need to focus on the papers in training and testing dataset.

4.3 BERT-based Method

In this section, we will introduce the method related to Bidirectional Encoder Representation of Transformer (BERT) [1]. BERT is a pre-training language representation which achieve state-of-the-art in many NLP tasks. With the characteristic of unsupervised, deeply, and bidirectional, BERT can be a perfect tool to generate document embedding with better performance. Moreover, it is really convenient to use BERT because it is also open sourced and support pytorch. However, training such a complex model comsumes lots of time. Fortunately, several pretrained models are also open-sourced. For example, bert-base-uncased model has 768 hidden layer number, and 110M parameters. On the other hand, bert-large-uncased model has 1024 hidden layer number, and 340M parameters. With pretrained models, we can easily encode a document using the output of the encoder.

Assumed we have the abstract sentence list where each element is the sentence of the abstract, we have to concatenate these sentences with [CLS] token added at the beginning and [SEP] token added between these sentences and at the end of the abstract. We then import the tokenizer from pretrained model.

The tokenizer will split the origin abstract into small pieces of token and convert them into corresponding id. After receiving the id list, we can load it into our pretrained BERT model to obtain the final embedding vector. In this work, we first convert all the training data into BERT embedding vector, and then use them to train the simple neural network afterwards.

The classifier we use here is a single hidden layer neural network with hidden neuron number equals to 512. We use adam optimizer with learning rate equals to $2 \cdot 10^{-5}$. The loss during the training process is as follows:

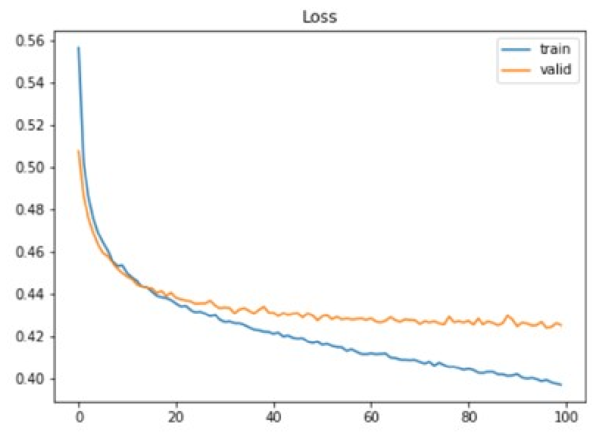


Figure 7: The loss during training BERT with base_uncased pretrained model.

The best F1 score on validation data is *0.6955*.

Although the BERT pretrained model has good performance on the NLP tasks, it may not have best performance on scientific task such as paper category classification. Therefore, Beltagy *et al.* [2] proposed a pretrained model, SciBERT, based on original BERT. The biggest difference between the origin BERT and SciBERT is that SciBERT is trained on a large corpus of scientific text, such as papers on arXiv. Therefore, although they have similar vocaulary size, the overlap vocabulary are only 42% of the size. We use the pretrained model of SciBERT and leave other settings as same as BERT. The loss curve during the training process is as follows:

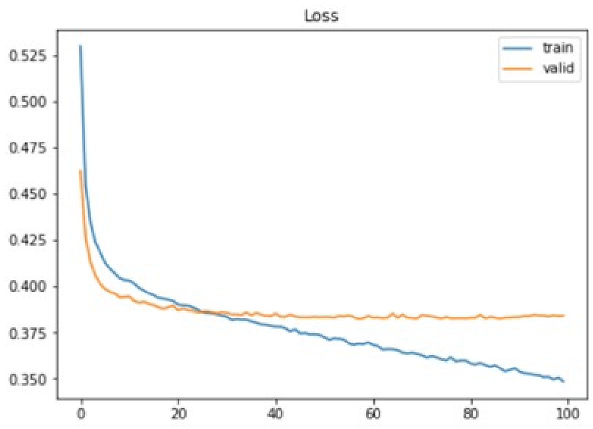


Figure 8: The loss during training SciBERT with scibert_scivocab_uncased pre-trained model.

Comparing to basic bert model, the F1 score of sciBERT model is *0.7215* which is more than *2%* higher.

5 Graph + Text Embedding

After obtaining the embedding from graph-based and text-based approaches, we find out that the performance of graph-based approaches have a huge performance gap compared to text-based approaches. Therefore, we come up with another idea, can we obtain another kind of embedding by joint considering text and graph information? Of course, this concept has been proposed in Text-associated DeepWalk (TADW) [3], which incorporates text features of vertices and graph representation learning. By adopting the concept in this paper, we utilize the extracted embedding from Doc2Vec with constructed category graph for further embedding extraction. The results of F1 score are shown in Table. 3.

Table 3: The F1 score of TADW

	All-Abstract-64	Train/Test-Abstract-Title-64
Feature	0.6202	0.6259
Category Graph	0.5299	0.5299
TADW	0.6298	0.6271

From Table. 3, we can find out that by joint considering graph and text information, the F1 score can be further enhanced.

6 Other Techniques

In this section, we will describe the other techniques to boost the performance of our prediction model. Although good model, nice preprocessing, and well-tuned hyper-parameters matter, there exist other method to enhance the overall performance. We will discuss each technique in the following sub-sections.

6.1 Regularization

In machine learning algorithm, it is important to avoid over-fitting. If a model can reach low loss on the training corpus, it maybe have much higher loss on validation and testing corpus. In order to avoid such situation, several techniques are proposed, such as adding dropout layer in neural networks, or adding regularization term in the loss function. We do both of them in our work. However, we want to discuss more on the regularization term. In our experiment, we apply two kinds of regularization, L1 regularization and L2 regularization. L1 regularization is easier to implement using weight_decay field in adam optimizer.

While implementing L2 regularization, we need to add another nn module of pytorch to trace the value of the neural network weight and add it to the total loss function as following:

$$Total\_Loss = BCELoss + \lambda \cdot L2\_Loss(weight) \tag{1}$$

where λ is the parameter that control the weighted of the regularization term. In our experiment, we choose $\lambda = 0.012$ and plot the loss during the training process as follows:

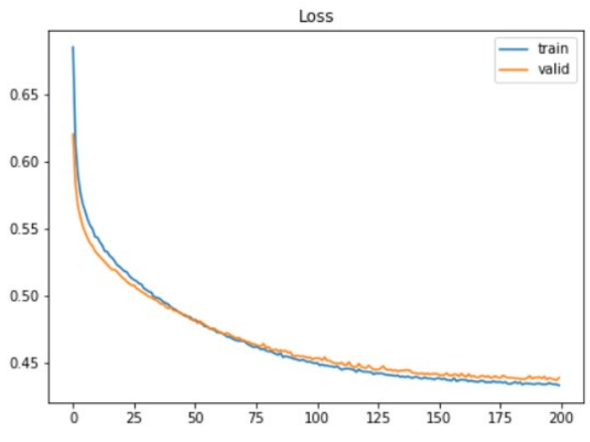


Figure 9: The loss during training SciBERT with L2 regularization term.

Comparing to Fig. 8, while the epoch number increase, the gap between the trainig loss and the validation loss does not increase in the SciBERT model with regularization term. Therefore, we can confirm that our model will not meet harsh over-fitting condition, and we can train our model on all training data without validation set, which may boost the F1 score due to the larger training data size.

6.2 Specific Output Threshold and Label Correction

Observing the task, we can find out that when calculating the output of a paper, if the first three categories are both 0, then the last category will be 1. Therefore, when we obtain a four-dimension output vector from the model, we preserve the first three dimension and calculate the label of each field. If all of them equal 0, we set the forth field to be 1, which is the output correction method. Moreover, because the output of the model is its probability of each category, the best threshold of each category may not be all 0.5. Therefore, we decide to determine the best threshold for each category using grid search method. The search range is [0.3,0.45] with interval equaling to 0.005. The effect of dynamic thresholding can be observed in the following figure:

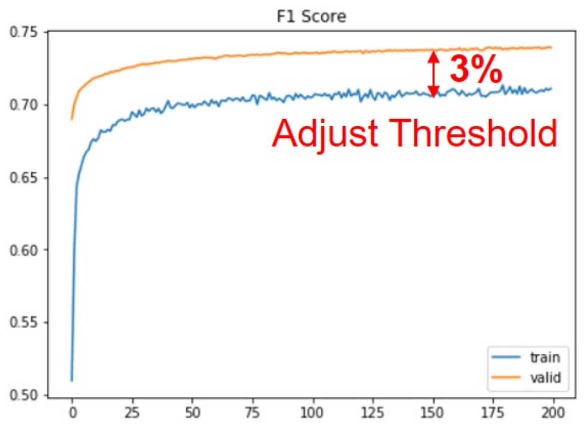
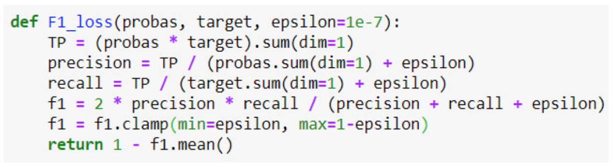


Figure 10: The F1 score during training SciBERT with L2 regularization term.

The effect of dynamic thresholding can boost the F1 score on validation set for about 3%. Therefore, we apply the dynamic thresholding and the label correction on our final model.

6.3 F1 Loss

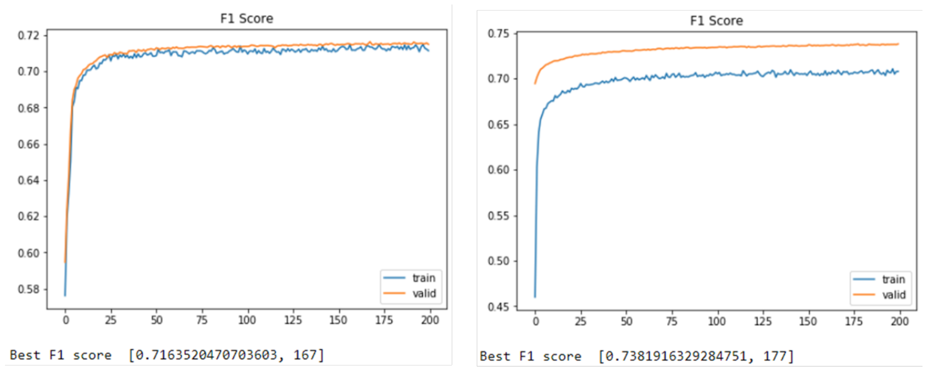
In this task, the evaluation metric is F1 score. However, the utilized loss function is binary cross-entropy, which may not be suitable for this task. Therefore, we try to define F1 loss function for the training process as shown in Fig. 11.



```python
def F1_loss(probas, target, epsilon=1e-7):
    TP = (probas * target).sum(dim=1)
    precision = TP / (probas.sum(dim=1) + epsilon)
    recall = TP / (target.sum(dim=1) + epsilon)
    f1 = 2 * precision * recall / (precision + recall + epsilon)
    f1 = f1.clamp(min=epsilon, max=1-epsilon)
    return 1 - f1.mean()
```

Figure 11: The defined F1 loss function for training.

And the Fig. 12 shows the F1 score during training with different loss function. We can find out that training with F1 loss has better train F1 score. However, the proposed specific output threshold in 6.2 can further enhance the validation F1 score for the training with BCE loss. Therefore, training with BCE loss still achieves better performance. For future work, may consider to add a parameter to adjust the percentage between these two loss function.



```
Best F1 score  [0.7163520470703603, 167]
```

```
Best F1 score  [0.7381916329284751, 177]
```

(a) Training w/ F1 loss (Best: 0.7164)

(b) Training w/ BCE loss (Best: 0.7382)

Figure 12: The F1 score during training for different loss function.

6.4 Ensemble

Another common adopted technique for performance improvement is ensemble, which takes the majority of prediction from different models. Therefore, for each prediction, we save hard prediction results, soft prediction results, and the thresholds for different categories. In this section, we propose two kinds of ensemble: hard and soft. For the hard ensemble, it directly utilizes the hard decision of prediction from different results. On the other hand, soft ensemble utilizes the soft probability prediction from different results, which can preserve more information. Besides, we will take the validation F1 score to weight the prediction results and thresholds. The results are shown in Table. 4 as below:

From Table. 4, the soft ensemble can achieve better F1 score than hard ensemble, which is reasonable.

Table 4: The F1 score of ensemble results

Ensemble Pool	Hard Ensemble	Soft Ensemble
Validation F1 Score ~ 0.7 **without Bert** (best TBrain: 0.6869)	0.6942	0.6955
Validation F1 Score > 0.735 (best TBrain: 0.7242)	0.7211	0.7225
Validation F1 Score > 0.74 (best TBrain: 0.7242)	0.7216	0.7241

7 Details of Best Performance Model

Although we propose lots of method to improve our model, the best model we obtained only combines some of them. Therefore, we will describe our best method in detail in this section.

First, after preprocessing of training, validation, and testing data, we choose title, abstract, and category to further analyze. First, we concatenate title and abstract and transform it into a DocEmbed vector using Doc2Vec. In addition, we process the category graph and DocEmbed using TADW to obtain CateEmbed. Finally, we concatenate title and abstract and load it into SciBERT model to obtain SciBERTEmbed. After processing each paper, we concatenate the DocEmbed, CateEmbed, and SciBERTEmbed to represent the paper. The final dimension of the vector is equal to 864.

Because dropout and L2 regularization will prevent our model to be overfitted, we do not split the task2_trainset.csv into training and validation set. We train our neural network using all training data. The neural network classifier has one hidden layer, which has 512 hidden neurons. We train our classifier using Adam optimizer and BCELoss from pytorch library. Moreover, we set the λ equals to 0.012. We train our classifier for 200 epochs. After training, we apply dynamic thresholding, and find the best threshold for the first three fields to minimize the F1 score on the training data. After training the classifier and deciding the threshold, we apply the whole system on test data and get the submission result.

The loss and F1 score during training is as follows:

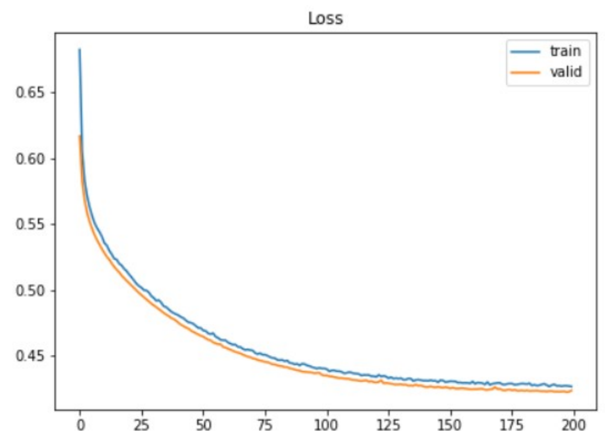


Figure 13: The loss during training process on best model.

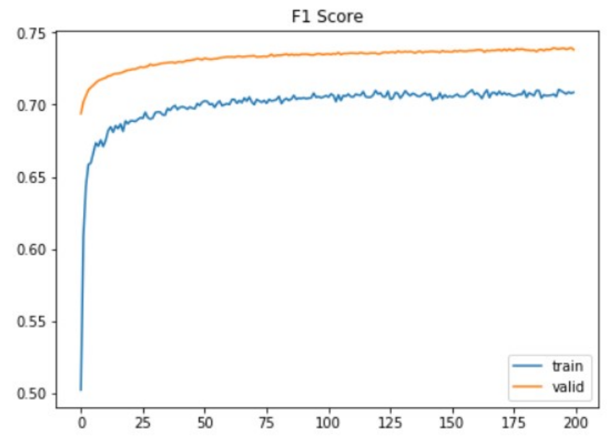


Figure 14: The F1 score during training process on best model.

The best F1 score happens at the 198 epoch, and the best f1 score is 0.7394. We then upload the public test answer and get 0.7242 on the F1 score.

8 Conclusion

In this task, we mainly extract two kinds of embedding, graph-based and text-based, from the available dataset. In the graph-based embedding, we obtain three different kinds of embedding from citation graph, author graph, and category graph, respectively. However, the performance of graph-based approaches can't achieve comparable performance as text-based approaches.

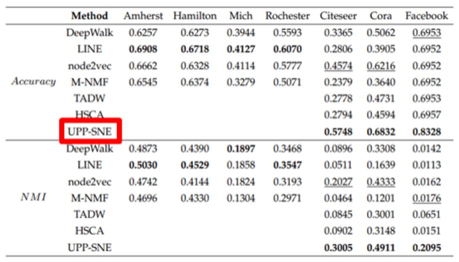
On the other hand, the state-of-the-art BERT [1] is the key to outperform the baseline model provided by TA. Furthermore, with similar corpus as this task, SciBERT [2] can achieve even better performance than BERT [1]. Besides, the adopted L2 regularization is important for effectively avoiding overfitting in this task. Due to the data imbalance, the best threshold for each category may not be 0.5. By applying dynamic thresholding technique, the performance can be improved.

With the best model, we can reach *0.7394* F1 score on the validation set and *0.7242* F1 score on the public testset.

9 Future Work

For the future work, we list some working items, which may be helpful for further improvement of performance as below:

- Take training data from another task (Abstract Labeling)
- Take BERT embedding results as TADW initial feature
- Try UPP-SNE which has significant performance improvement as shown in Fig. 15 [4]
- Finetune SciBERT model using our corpus
- Adaboost multi trained model



	Method	Amherst	Hamilton	Mich	Rochester	Citeseer	Cora	Facebook
Accuracy	DeepWalk	0.6257	0.6273	0.3944	0.5593	0.3365	0.5062	0.6953
	LINE	**0.6908**	**0.6718**	**0.4127**	**0.6070**	0.2806	0.3905	0.6952
	node2vec	0.6662	0.6328	0.4114	0.5777	0.4574	0.6216	0.6952
	M-NMF	0.6545	0.6374	0.3279	0.5071	0.2379	0.3640	0.6952
	TADW					0.2778	0.4731	0.6953
	HSCA					0.2794	0.4594	0.6957
	UPP-SNE					**0.5748**	**0.6832**	**0.8328**
NMI	DeepWalk	0.4873	0.4390	**0.1897**	0.3468	0.0896	0.3308	0.0142
	LINE	**0.5030**	**0.4529**	0.1858	**0.3547**	0.0511	0.1639	0.0113
	node2vec	0.4742	0.4144	0.1824	0.3193	0.2027	0.4333	0.0162
	M-NMF	0.4696	0.4330	0.1304	0.2971	0.0464	0.1201	0.0176
	TADW					0.0845	0.3001	0.0651
	HSCA					0.0902	0.3148	0.0151
	UPP-SNE					**0.3005**	**0.4911**	**0.2095**

Figure 15: The performance of different approaches [4].

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[2] Iz Beltagy, Arman Cohan, and Kyle Lo. Scibert: Pretrained contextualized embeddings for scientific text. *CoRR*, abs/1903.10676, 2019.

[3] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. Network representation learning with rich text information. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[4] D. Zhang, J. Yin, X. Zhu, and C. Zhang. Network representation learning: A survey. *IEEE Transactions on Big Data*, pages 1–1, 2018.