Project Report of Drug Molecular Toxicity Prediction

Diao Yijia
Department of Computer Science
Shanghai Jiao Tong University
diao_yijia@sjtu.edu.cn

Abstract

This is the project: Drug Molecular Toxicity Prediction report of CS410: Artificial Intelligence. Nowadays, real-world clinical trials for assessing drugs are extremely time-consuming, thus a computational drug molecular toxicity assessment method is necessary to be developed to quickly test whether certain chemicals have the potential to disrupt processes in the human body of great concern to human health. To solve the problem of classifying drug moleculars, I propose a GCN-liked model to do the binary classification task, especially on the course provided dataset. The motivation of this model is to extract more effective information from pysmiles. The performance of the model performs a score of 0.864 AUC test, and is more effective compared to the GCN model.

Keywords Binary Classification · Neural Network · GCN · SMILES · Graph Computation

1 Introduction

Almost all people are exposed to different chemicals during their lifetimes through different sources including food, household cleaning products and medicines. However, in some cases, these chemicals can be toxic and affect human health. As a matter of fact, over 30% of drugs have failed in human clinical trials because they are determined to be toxic despite promising pre-clinical studies in animal models. Consider real-world clinical trials for assessing drugs are extremely time-consuming, it is ideal if a computational drug molecular toxicity assessment method can be developed to quickly test whether certain chemicals have the potential to disrupt processes in the human body of great concern to human health.

Deep neural network has become a hot research topic in machine learning in recent years. Compared to other methods, deep learning has shown its advantages in handling large amount of data and achieving better performance. [1] Graphs are a kind of data structure which models a set of objects (nodes) and their relationships (edges)[1]. In this individual project, I use the given dataset of drug molecules with their SMILES[2] expressions and the binary labels indicating whether one drug molecule is toxic or not. I have implemented a GCN-based (Graph Gonvolutional Network) network to do the binary classification task, using pysmiles[3] and networkx[4] to extract preliminary features of SMILES expressions, based on learned knwoledge of PyTorch Package.

2 Model

In this section, I will introduce my GCN-based model in detail.

[1] From the CS410 Project 1 Specification (2020 Autumn) File.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

2.1 Feature Extraction and Adjacency Construction form SMILES

In this part, the model extracts features and construct adjacent matrix from the chemical expression in the form of SMILES[2]. Although the one-hot format of SMILES is also available in the dataset, this model does not use it.

This part is based on pysmiles[3] and networkx[4].

Using pysmiles, the molecules are depicted as Networkx graphs, with each nodes have a number of attributes. After observing a certain scale of each molecule's attributes, 'element','aromatic','hcount' and 'charge' are useful for extracting the feature from the SMILES in the dataset. Other attributes cannot be used since the SMILES's information is limited: it did not provide the 'isotope' and 'class' of each element in the graph, and I also did not add those information to this model. Therefore, the feature matrix H 's feature dimention has 4 attributes. The feature matrix is sparse, and thus we cannot use only one GCN layer to extract the

The adjacent matrix of each molecule is derived from the pysmiles's pretreatment. The model change the graph to a numpy[5] array using networkx's method.

The feature extraction and adjacency construction part is implemented in data processing file(dataproc.py).

2.2 GCN Layer

The GCN layer is based on the work of Kipf and Welling[6], and re-constructed to adapt the specific task from Kipf's implemention in [7].

The key formula expression used in the model is:

$$O^{(l)} = (\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}})H^{(l)}W^{(l)} \tag{1}$$

In which $H^{(l)}$ stands for and the feature matrix and it's the input of layer l, $A = \tilde{A} - I$ is the adjacency matrix of the graph, $\tilde{D}$ is the diagnal degree matrix of the graph, $W^{(l)}$ is the training weight matrix, $O^{(l)}$ is the output of layer l. After using activation function of the output layer, we can get a new feature matrix $H^{(l+1)} = \sigma(O^{(l)})$, as the input of $(l+1)$ GCN layer, or other type of layer.

As the input and output are all feature layer, when adjusting GCN, we need to care about the number of features of each layer.

2.3 GCN Model's Structure

I first implemented a 3-layer GCN model. The parameter of this model is in Section3.1

2.4 GCN-liked Model

Since the performance of this 3-layer GCN model is no better than the 2-layer CNN model provided by the TA, I make a roughly tricky modification of the input and output of GCN layer, to build a 2-layer GCN-liked model.

The first layer is a normal GCN layer, with the expression of Eq1. Then the output: $In^{(2)} =$ Relu($(O^{(1)})$) of the GCN layer won't be the input of next layer's feature. It will work as the input of next layer's "adjacency", the formula is:

$$O^{(2)} = In^{(2)}H^{(1)}W^{(2)} \tag{2}$$

It's actually exchange the "role" of adjacency matrix and the feature matrix.

The motivation of doing this is that, the adjacency matrix and the feature matrix has **no essential difference**. They are all constructed from the pysmile implemention and networkx's transformation which I do not actually know what they have done. Thus I choose to take them fairly, treat both of them as the feature of the SMILES with different expression. Also, the choice to keep the $H^{(1)}$

as the input of the second layer can extract more useful information from it, as I have filtered the useless attributes in the graph.

The other reason is, when processing data, I choose to refill the quantity of the adjacency matrix and the feature matrix to avoid the disalignment fault when running with CUDA. The refill value is all choosen 0, which cause an information confusion. As I observed, the attribute 'charge' 'aromatic' is much more sparsed, add more 0 to the attributes will only cause more imformation mistaken for the neural network.

The implemention of the GCN-liked model is in myGCN.py. Note that I implement the calculation of $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ in dataproc.py, not in GCN layer.

3 Experiments & Discussion

In my experiments, I only use the "/train/name_smiles.txt" for training. The loss function for both models is "BCEWithLogitsLoss()". Since adding "batch" will lower down the performance, I choose to use CUDA for traning on the server.

3.1 GCN Model

The paremeters in GCN model are:

Table 1: Parameters of GCN model

Parameter	Value
Learning rate	0.0005
Epoch	20
Batch	1
Hidden feature number 1	16
Hidden feature number 2	8

With these parameters, I get the best performance result of 0.84212 in my experiment.

Change the learning rate to 0.002 will have a stable performance ranging from (0.835-0.840) within 50 epochs.

Here is the discussion of the result of the model. This probably is not the best performance of this network, since I have only tried 2-layer GCN, 2 group of hidden feature number, 3 values of learning rate, all within 50 epochs.

The problem may lies in the data processing. Through the network and data processing, I did not use any normalization for any input or output. This is not a good thing for GCN, because for sparse matrix, any imbalance of the input can cause a large difference in the trained model.

3.2 GCN-liked Model

The paremeters in GCN-liked model are:

Table 2: Parameters of GCN model

Parameter	Value
Learning rate	0.001
Epoch	7
Batch	1
Hidden feature number	2

With these parameters, I get the best performance result of 0.86436 in my experiment.

Here is the discussion of the result of the model. This probably is the best performance of this network, since I have done a lot experiment of the parameters of this model.

This model has the same question as GCN model, they both does not take normalization operation.

3.3 Comparison with the two models

First, the better performance of GCN-liked model has proved that my idea in Section 2.4 is reasonable to some extent.

However, when adjusting the parameters of these two networks, they have some similar properties, but with some special properties that has much difference. GCN model performs better when using large hidden feature number, but GCN-liked model performs better with smaller hidden feature number than the input feature number. The possible reason is, GCN aims at extracting features from the adjacent graph and its node's features. My GCN-liked model mix the boundary between the adjacency and feature (and actually they are mixed, as explained in Section 2.4). Our target is to do binary classification, thus less hidden feature number can concentrate the features.

4 Conclusion

In this project, I have built a 3-layer GCN and construct a 2-layer GCN-liked model. The motivation of using the second model is to extract more effective information from pysmile-processed SMILES. The experiment result proves that the GCN-liked model is effective.

The future work should be using the embedding method to process the initial features of pysmile-processed SMILES, in order to solve the problem of feature mixure with adjacency. I also want to try to apply BERT(Bidirectional Encoder Representation from Transformers) for this binary classification task.

5 Acknowledgments

Thank our teacher for providing this chance for us to grade our model in the competition.

Thank TA for his project demo, which leads me to neural network programming.

Thank Wei Yifei for telling me pysmiles is available for this project.

Thank Hou Shengyuan for telling me RDKit is available for this project, although I do not use it.

Thank Huang Zhongyu for telling me the official examples of pytorch may help a lot in programming.

Thank Gu Ruiqi and Fang Ya, they teach me about some details of the use of pytorch. Although most details are not used in the final submission, they do help me a lot with the programming. And especially thanks to Gu Ruiqi, when we discussing the details of GCN layer, I came out the idea of this GCN-liked model (sorry for not sharing this part with her).

References

[1] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[2] O. Community. (2020). "OPENSMILES," [Online]. Available: `http://opensmiles.org/` (visited on 12/27/2020).

[3] the Python community. (2020). "pysmiles 1.0.1," [Online]. Available: `https://pypi.org/project/pysmiles/` (visited on 12/27/2020).

[4] N. developers. (2020). "NetworkX," [Online]. Available: `https://networkx.org/` (visited on 12/27/2020).

[5] Google. (2020). "NumPy," [Online]. Available: `https://numpy.org/` (visited on 12/27/2020).

[6] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[7] T. N. Kipf. (2018). "pygcn,Graph Convolutional Networks in PyTorch," [Online]. Available: `https://github.com/tkipf/pygcn` (visited on 12/27/2020).