COMP90051 Project 2 Report

Group 9
Group Members: Cenxi Si, Keyi Zhang, Zihan Xu
GitHub Repository Link: https://github.com/cenxi-si/COMP90051_Project2_Group9

1 Project Overview

In this project, we are going to predict the prolific authors of a given document. There are two provided datasets in JSON format for training and testing. Five features are included in training data: *author IDs, year of the paper was published, publication venue, paper title,* and *abstract*. The *Authors* field, ranging from 0 to 21245, represents a collection of authors. The first 100 authors, identified as prolific authors, are the prediction target. Year and venue are mapped to a unique integer. The title and abstract are sequences of words, each word is mapped to an index in the range of 1 to 4999. This project aims to determine who of a group of 100 well-known authors contributed to each of the 800 papers in the test set. The right response might be none, one, or many of these writers.

2 Approaches

2.1 Data Pre-processing

Training Set Optimization

We found an uneven distribution of prolific authors. In the training set after splitting, there are 17409 papers without prolific authors and only 7094 papers with prolific authors. So we randomly sampled papers from those without prolific authors to get the same amount of papers with/without prolific authors for even distribution. To validate, we tested the model trained by adding or subtracting 25% of the samples with empty labels respectively, and the results confirmed that the model trained by the uniform distribution had the highest accuracy.

Splitting Labels in Training Set

Since the author field in the training data set is a combination of prolific authors and coauthors, we need to split them into two fields to separate the feature and target we should predict.

2.2 Techniques

One-Hot Encoding

Many algorithms could not directly act on label data. They demand that all input and output variables be numbers. Thus, the categorical data must be transformed into numerical data. In this project, since the coauthor has 21147 categories and the target prolific author has 101 categories with -1 for none, some encoding methods are necessary to be used. One-hot encoding (Brownlee, 2017) is a method to convert categorical features into binary classification using a vector. 1 denotes existence, whereas 0 denotes nonexistence. The length of this vector is the same as the total number of categories in features. For example, if the paper is written by coauthors 10 and 20, the index of 10 and 20 in the vector would be changed to 1 and others remain 0. After applying one-hot encoding on coauthors and prolific authors, for each paper, the coauthor becomes a 21147-length-list and the prolific author turns into a 101-length-list. Both of them only have integers 0 or 1. For other variables, we also transform them into binary classification format. One-hot encoding is easy to implement and could make the training faster. However, one-hot encoding is difficult to represent the relationship between words within the title and abstract. Therefore, another transformation method called word2vec embedding will be introduced next.

Word embedding

Word2vec (*Word2Vec Model — Gensim*, 2022) is widely used in Deep Learning, especially in Neural Networks. It takes a large amount of raw text that has not been annotated to automatically learn the relationships between words. Word embedding considers context and assigns words in a phrase with a similar meaning or effect a comparable value

for a certain attribute. The result consists of vectors with remarkable linear relationships, one vector for each word. Also, it may retain the sentence context from the words, and after training on a large data set, it can even identify words from sentences that are not represented as vectors. Here in our dataset, the *title* and *abstract* are sequential texts, there is a strong relationship between a word and the word before/after it. Hence, simply using one-hot encoding, especially on *abstract*, is not enough. After training on the entire dataset with title and abstract, we could use this pre-trained model on the unlabelled dataset. However, it would take longer to operate.

Neural Network-Pytorch

To build a neural network as our model, PyTorch is a good choice. We chose PyTorch over Tensorflow because it is a new-generation framework written entirely in Python and has a higher level of usability. This means that small projects are easier to start and debug. Neural networks can be used from the nn.torch package from PyTorch (*Neural Networks — PyTorch Tutorials*). In a simple feed-forward network, it takes the input, feeds it through several layers one after the other, and then finally gives the output. The following describes a typical neural network training process:

I. Define the neural network using a few parameters that can be learned (or weights)
II. Iterate every tensor-format input from the dataset
III. Process the input through the built network with defined layers
IV. Calculate the loss after fitting the model to see the performance
V. Compute the f1 training score to evaluate the prediction accuracy
VI. Propagate backward to the networks' parameters
VII. Update the network's weight

However, a traditional neural network model could not deal with the connection of the previous and afterward words in a text. Recurrent neural networks (Graves, 2015) could manage this problem. They use loops in networks to hold the information and can also be considered duplications of a single network by passing information to the next. A connection between the previous and current words could be built. RNN is good at using previous information to perform the current job. When we need to consider a longer context, RNN could not perform well enough. Long short term memory networks (LSTMs) can improve this drawback with the ability to pick up on long-term dependence. It contains duplicated networks with various structures and performs interaction among them. In this project, we found that LSTMs could perform better than traditional neural networks.

2.3 Combination of Features

To improve model performance, we tried different combinations of features and the following are all working possibilities ordered by accuracy (best to worst):

1) Coauthor with one-hot encoded using LSTMs
2) Title + coauthor with one-hot encoded using LSTMs or traditional Neural Networks
3) Title + abstract + coauthor + year + venus with all one-hot encoded with Multi-label KNN
4) Title or abstract with word2vec embedding with LSTMs or traditional Neural Networks

2.4 Software Setup

The rate at which deep learning trains a model depends on the computer configuration. GPUs are faster because they use parallel processing, dividing tasks into smaller subtasks that are distributed across a large number of processor cores in the GPU. This results in faster processing of specialised computing tasks. As a result, our approach provides two options: CPU and GPU. In GPU mode, we use the CUDA computing framework, which is to call cuda() on tensors to instruct PyTorch. So that it can significantly reduce the time required to train the model. While in CPU mode, the training time is relatively long.

2.5 Final Approach

After comparisons, we decided to use coauthors with one-hot encoding alone to predict prolific authors. The model we chose is LSTM networks. As we discussed before, LSTMs could perform better than traditional neural networks due to the ability to pick up long-term dependence. When we use coauthor and any other features like title and abstract, the f1

score from the training set would approximate 1 which causes overfitting. The reasons would be the model learns too much on the training dataset with a large number of iterations and the combination of one-hot encoded features usually becomes a vector with an extremely long length which leads the model too complex. In addition, we choose BCELoss to be the loss function and use the sigmoid activation function to ensure our predictions from the model are between 0 and 1. Since the Cross-entropy loss function is mainly utilised for binary and multi-classification issues, it can alleviate the problem of the squared loss function's weight being updated too slowly. Our task is a multi-label classification which means the target could assign to one or more or none classes. Neural networks could work better in multi-label classification because the number of target labels can be present as the number of nodes from the output layer. It could detect the possible connections between predictors and train more complex correlations among features. Moreover, the possible reason for the bad performance on word embedding might be the mapping technique on the title and abstract. Since it might split a word into 2 or more parts, it would limit the performance of word2vec.

With this model, we got an accuracy of 0.50234375 on our validation set and an F1 score of 0.53452 in the competition. Although the performance is not bad, we found it hard to improve further. One possible reason might be the unbalance of empty prolific authors in the training set and testing set. Although we remove part of the papers with empty prolific authors from the training set, it also decreases the size of training data at the same time.

2.6 Other Alternatives

Link Prediction

One way to predict prolific authors is to find whether there is a strong link between a prolific author and a coauthor. We constructed a network using the networkx library. Each node is an author and an edge between two nodes is one collaboration of two authors. The weight of an edge is the number of collaborations. However, we are able to construct a network of existing collaborations, but it is hard to make predictions on a given new paper.

Multi-label kNN

Since the target prolific author is multi-labelled, a traditional knn classification could be adapted. It uses the k-NearestNeighbours to find the kth nearest samples to the paper in the testing set and applies Bayesian inference to choose the assigned labels. However, since the size of the training data is large and the target contains 100 classes, the model using knn would become complex. The traditional machine learning model makes decisions from the knowledge learned from data (Goyal, 2020), but neural networks could improve themselves from the mistakes they made. Neural networks also use graphs of neurons and a variety of methods to model data without requiring human intervention. Thus, we choose neural networks to fit this training set.

References

Brownlee, J. (2017, July 28). *Why One-Hot Encode Data in Machine Learning?* Machine Learning Mastery. Retrieved October 19, 2022, from https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

Goyal, K. (2020, February 13). *Machine Learning vs Neural Networks: What is the Difference?* upGrad. Retrieved October 20, 2022, from https://www.upgrad.com/blog/machine-learning-vs-neural-networks/

Graves, A. (2015, August 27). *Understanding LSTM Networks -- colah's blog*. Colah's Blog. Retrieved October 20, 2022, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Neural Networks — PyTorch Tutorials 1.12.1+cu102 documentation. (n.d.). PyTorch. Retrieved October 19, 2022, from https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

Word2Vec Model — gensim. (2022, May 6). Radim Řehůřek. Retrieved October 19, 2022, from https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html