111-1 Fall Applied Deep Learning

Team 13 Final Project – Hahow Course Prediction

R11725053 資管碩一 朱育萱

R11944022 網媒碩一 吳雲行

R09944076 網媒碩二 陳宏昇

Abstract

With the rapid development of deep Learning, a lot of application utilizes this technique to become more convenient and more powerful. The algorithm of providing personalized advertisement, commodities, and other preferred items for each client has always attracted the attention of commercial organizations because of the potential substantial benefits. However, the difficulty of developing a fabulous recommendation model is the need to handle categorical feature and the raw text data. So in this Hahow course prediction task, it is necessary to find a proper way to represent the preference of users and the characteristics of different courses into numerical data either in the traditional way or the approach of deep learning. In addition, how we merge the features between users and courses is also the key component for this task.

Related work

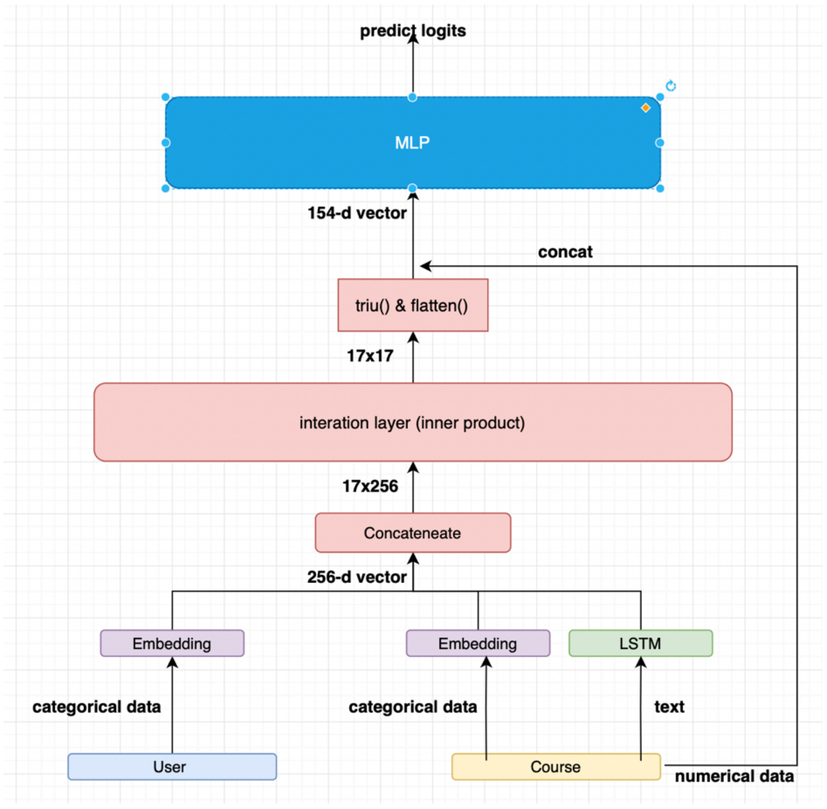
Traditionally, most corporations develop their own recommendation algorithms with the combination of the domain knowledges of experts, the characteristics of their products and the analysis of target clients. For example, most recommendation algorithms use collaborative filtering [1] to represent the relation between users and courses with matrix and find their similarity with this matrix. Nowadays, with the help of easy access to plenty of data, good personalization predictions can be achieved by training a mighty deep learning model with the previous data.

Proposed methods

1. Deep Learning Recommendation Model (DLRM)
   Motivation

DLRM uses the way of embedding to encode the input data and use the interaction layer to combine the information, which seems a promising solution to this task. What's more, DLRM's concept is straight and simple, so we use it at first to acquire the inspiration for the following methods design. Architecture

We refer to the concept of Deep Learning Recommendation Model (DLRM) [2] to design our model. First, we pass the categorical data of users and courses into individual embedding layers and convert them into 256-d vectors. For the text data such as course introductions and topics, we use Long Short-Term Memory (LSTM) [3] to encode them into 256-d vectors as well. After that, we implement the interaction layer where doing the inner product for each vector pair and flatten the results. Subsequently, we concatenate the resulting flattened vector and the raw numerical feature such as course price. Finally, pass the vector into Multi-layer perceptron (MLP) layers and output the predicted logits probability.



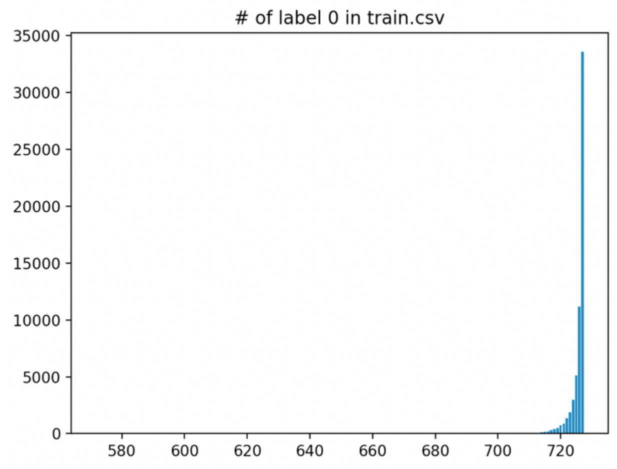
Techniques

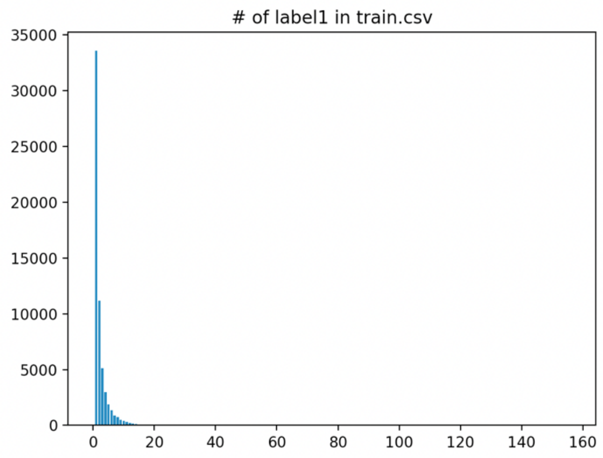
With the research of the data, we found that there are three main issues in the data.

(1) Infeasible training time. In training data, there are 59737 and 728

data for users and courses, respectively. So if we train the model with each user-course pair, we have 43488536 training data in total, which is time-consuming.

(2) Imbalanced data. In training data, most users just buy 1~20 courses and remain the left 700~727 lessons unbought as shown





(3) No label 0 (unbought course) data. Actually, we only know which courses the users bought and which courses aren't bought at that time the data is collected (pseudo label 0). But we don't know which courses the users won't buy forever.

To deal with the issues, we use the method of sampling for (1) and (3). We sample from the unbought courses with the size of label 1 (the number of bought courses) for each user in each epoch to shrink the size of training data and the training time. And we assume that the labels of sampled courses are 0 because we believe that the number of actual label 0 is far larger than label 1. To overcome the issue of imbalanced data, we use weighted binary cross entropy as the loss function.

Results

	original	weighted BCE

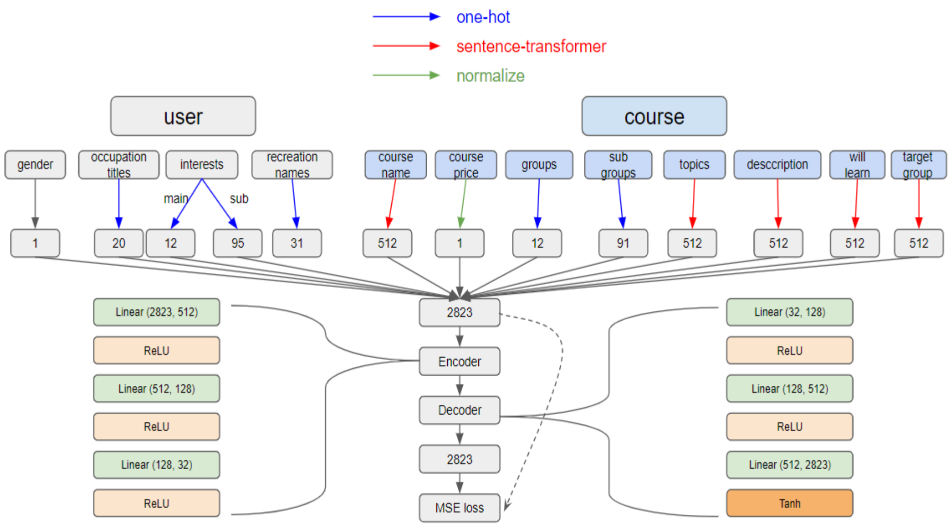
MAP@50 of val	0.0281	0.0715
MAP@50 of test	-	0.03589

2. Autoencoder [4]
   Motivation
   First, because it is hard to validate the correctness of assumption we proposed in the previous model, we want to find a way to overcome the problems of infeasible training time and pseudo label 0 without sampling. Unsupervised learning autoencoder is a proper way to get rid of the issue of labels. What's more, if we only train the model on the label 1 (bought) courses, the number of training data hugely decreases.
   Second, since we only train on the user-bought course pairs, the user-unbought course pairs will be the outliners for the autoencoder. As a result, we would like to utilize the ability of anomaly detection of autoencoder to find the score between users and scores. In short, the lower the reconstruction loss is, the higher the user-course score is.
   Architecture
   - Convert the data of user and course into one-hot vectors for the categorical data. And encode the text data into 512-d vectors whose values are in [-1, 1] with pretrained sentence transformer. For the numerical data, we normalize it into the range [0, 1].
   - Concatenate them into a 2823 vector for each user-course pair.
   - Pass the vector into the encoder and decoder composed of several linear layers, ReLU layers, and a Tanh for convert the output into [-1, 1].
   - Calculate the reconstruct between the input and result.
   - Select the user-course pair with the lowest reconstruction loss as our predictions.

   Results

	Seen users	Unseen users

MAP@50 of val	0.06195	0.07980
MAP@50 of test	0.04077	0.05248

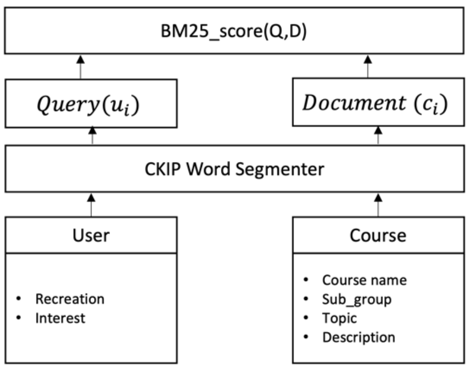


3. Collaborative Filtering

   Motivation

   The training and inference time of deep learning model is too long to fit this model. Even the autoencoder we just mentioned can shrink the training time, but it still suffers the long inference time. As a result, we consider to give up the DL-based framework.

Method 1



- Concatenate all the text and categorical data of users and courses.
- Use CKIP word segmentation to segment the text.
- Use BM25 score formulation to turn each usertext-course-text pair into score. The formulation is shown below

(usertext = query, coursetext = document):

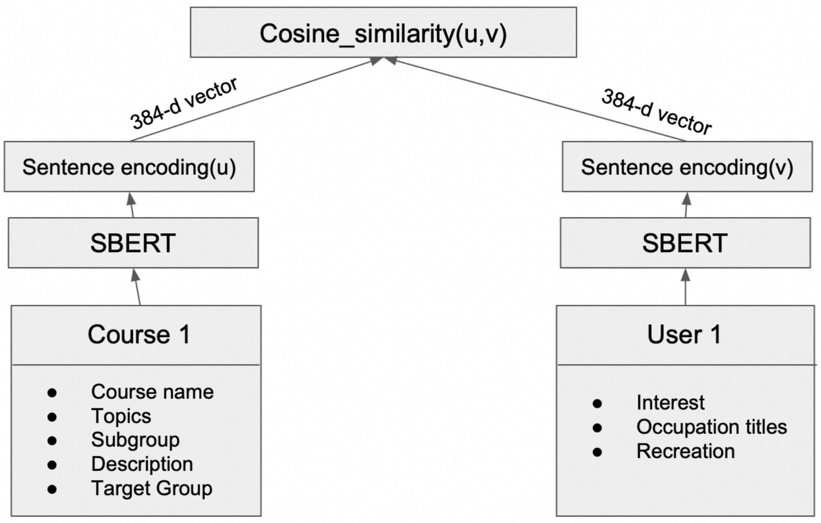
$$score(Q,D) = \sum_{i=1}^{n} IDF(q_i) \times \frac{f(q_i,D) \times (k_1+1)}{f(q_i,D) + k1 \times (1 - b + b \times |D|/avg(dl))}$$

where

1. $IDF(q_i) = ln\left(\frac{N-n(q_i)+0.5}{n(q_i)+0.5} + 1\right)$
2. $f(q_i, D)$ *is the occurrence of query token* q_i *in document D.*
3. *k1 is a constant in [1.2, 2.0], b = 0.75, avg(dl) is the average length of documents.*
4. *N = the number of documents*
5. $n(q_i)$ *= the number of documents containing the word* q_i

- Sort the scores and output the courses with the highest score as our recommendation.

Method2



- Concatenate all the text and categorical data of users and courses.
- Use sentence transformer SBERT instead of CKIP to encode the text into 384-d vectors because SBERT is a multilingual model.
- Calculate the cosine similarity of user-course vector pairs.

$$S_c(Q,D) = \frac{\sum_{i=1}^{384} Q_i D_i}{\sqrt{\sum_{i=1}^{384} Q_i^2}\sqrt{\sum_{i=1}^{384} D_i^2}}$$

- Sort the scores and output the courses with the highest score as our recommendation.

Method3

We consider the popularity of courses is the important factor. As a result, we calculate the # of bought counts of each course as the popularity score. And do the min-max normalization and the result is in [0, 1]. Finally, we calculate the final score with the weighted sum of popularity score and the score we output in the last method.

$$S_c(Q,D) = 0.7 * \frac{\sum_{i=1}^{384} Q_i D_i}{\sqrt{\sum_{i=1}^{384} Q_i^2}\sqrt{\sum_{i=1}^{384} D_i^2}} + 0.3 * popularity(D)$$

The weight 0.7 : 0.3 is the best setting we try.

Result

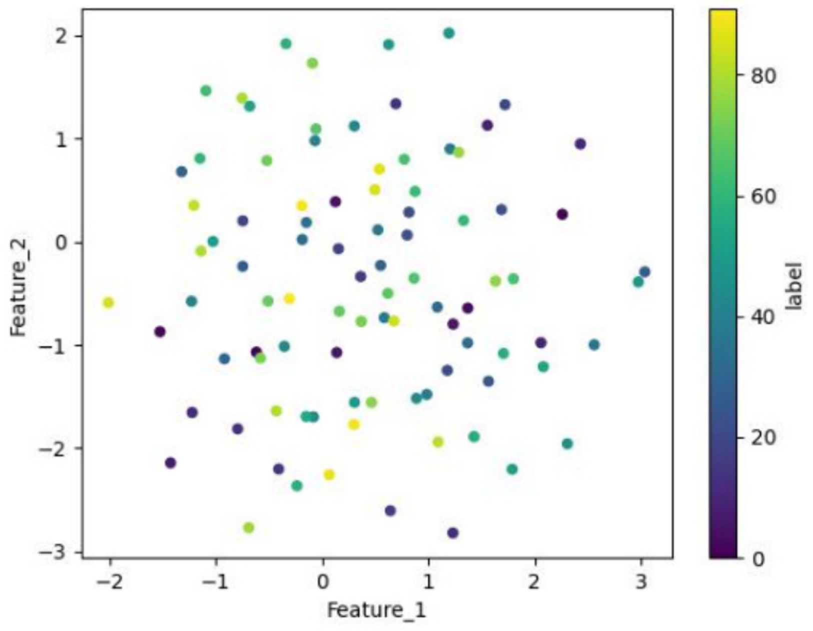
	BM25 with CKIP Segmentation	SBERT(cosine similarity)	SBERT with popularity
MAP@50 of val (unseen course)	0.01221	0.054579	0.0761352
MAP@50 of test (unseen course)	0.04481	0.05879	0.0668
MAP@50 of val (unseen topic)	0.225089	0.28387	0.2908595
MAP@50 of test (unseen topic)	0.25166	0.29862	0.30272

SBERT with popularity	Weight = 0.7 : 0.3	0.4 : 0.6
MAP@50 of val (seen course)	0.074229	0.072431
MAP@50 of test (seen course)	0.04415	0.0457
MAP@50 of val (seen topic)	0.26671	0.23516
MAP@50 of test (seen topic)	0.28724	0.25207

Experiments & Discusion

The validation and test result has already shown above. Here we do some visualization to analyze the result.

For model 1 DLRM, we find the result is bad because the training time is too long to select the proper parameter to fit the data. So we use t-SNE [5] to visualize the 92 output vectors of subgroup embedding layer. The result is the following figure:



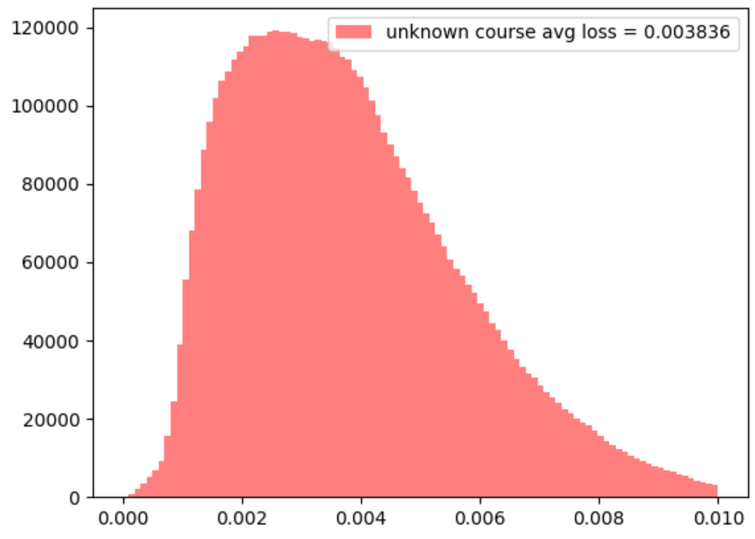
From this figure, we can find the embedding layer does not learn very well because it does not convert the similar subgroups into clusters. We think this main reasons are :

- We don't have enough time to train this model to fit so much

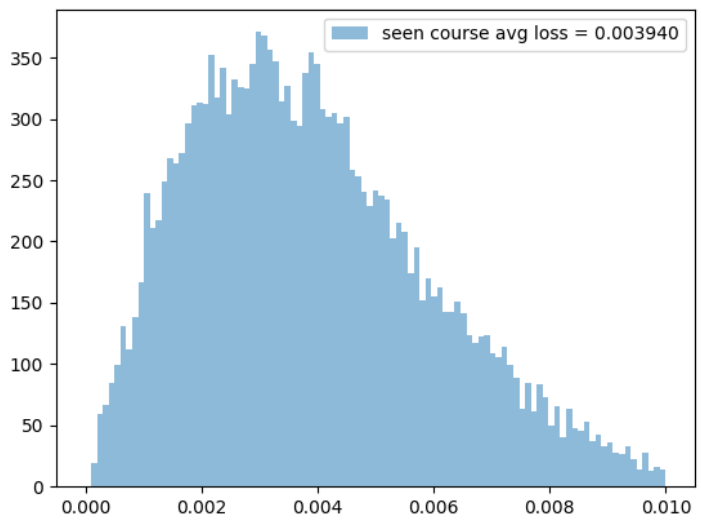
training data.
- The column is too many so that each embedding layer only learn a little representation.

We consider the better scenario is to train this embedding layer in advance. And freeze it in the following training. It may get better result and shorter training time.

The result of autoencoder is not good enough, either. Although the autoencoder model does not need any assumption and get more stable result, we still don't know whether the feature of positive pairs are similar and whether there is enough difference between the positive pairs and negative pairs for autoencoder to separate them. So we visualize the reconstruction loss as following figures :

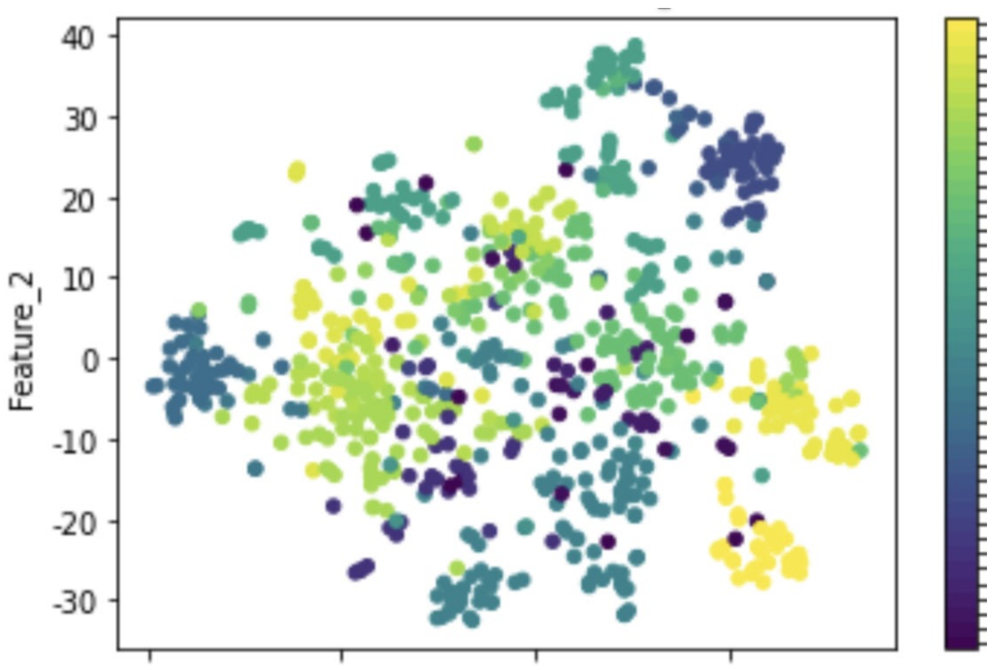


-



The first figure is the MSE loss distribution for user-unbought course pairs in val_seen.csv and the second is the MSE loss distribution for user-course pairs in val_seen.csv. From these two figures, we can find that actually the difference between bought and unbought courses is small.

The third model gets the best result. So we visualize the output of its SBERT transformer feature with t-SNE, too.



We can find that the output vectors are clustered well according to their groups. Therefore, the SBERT encodes the information well. So this model can get better results. In addition, its performance on topic prediction is better than many other teams, which shows it can really encode the group information, too. From these results and experiments, we know that it is more important to understand the characteristics of data before choosing the models. In addition, the simpler collaborative filtering combined with the sophisticated design can also get better results. The DL-based methods does not always fit any conditions.

Conclusion

Our team uses three kinds of methods for this task. The first one is DLRM, which has deep architecture and is powerful but heavily suffers from infeasible training and inference time and leads to a bad performance. The second one is autoencoder. The autoencoder successfully gets rid of the assumptions about labels. So its training is more possible and get better score than the first one. However, our experiment shows that it cannot really classify the unbought and bought courses from the reconstruction loss. The last one uses the traditional concept of collaborative filtering, and utilize the text encoder to calculate the similarity between users and courses. And its inference time just spends 3-5 minutes and get the best result. So we choose the third one as our best model.

Reference

[1] Ruisheng Zhang et.al., Collaborative Filtering for Recommender Systems, IEEE 2014
[2] Maxim Naumov et.al., Deep Learning Recommendation Model for Personalization and Recommendation Systems, IR 2019
[3] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.
[4] Dor Bank et.al., Autoencoders, ML 2020
[5] L van der Maaten et.al, Visualizing Data using t-SNE, 2008