2022 ADL FINAL PROJECT - HAHOW CHALLENGE

b09902016 鮑鈺文 b09902017 李安傑 b09902019 沈立程 b09902135 賴豐彰

{b09902016, b09902017, b09902019, b09902135}@csie.ntu.edu.tw

Abstract

For our final project, we participate in the Hahow Grand Challenge. In this challenge, we are provided data of users and courses, and a training dataset of users' course purchase records. Our goal is to accurately predict additional course purchases, and this is split into four tasks, each corresponding to predicting either the courses or subgroups of courses, and whether the users are seen in the training set. To tackle this problem, we employ and experiment with several methods including decisions trees, LightFM, neural factorization machines, multi-layer perceptrons, pre-trained NLP models with cosine similarity, and transformer encoders.

1. INTRODUCTION

Recommender systems are widely used in electronic commerce and social media services. The objective of such systems is to provide users with suggestions on items most relevant to them based on their preferences or past behavior. Machine learning techniques have given rise to significant progress in providing effective and accurate recommendations. Moreover, the utilization of pre-trained NLP models enable improved analysis of text data, which can be useful for such tasks.

2. RELATED WORKS

Approaches to building recommender systems can be classified as content-based filtering [1], which recommends items according to the characteristics of a user, or collaborative filtering [2], which is based on the users' past behavior such as previously selected items. Addressing such recommendation tasks typically requires applying methods of predictive analytics and machine learning. Some of the useful techniques include random forest [3] and LightFM [4]. Furthermore, recent advances in deep learning have brought about a variety of powerful methods which can be employed. Neural factorization machines [5] utilize the non-linearity of neural networks to improve the performance for prediction under sparse settings. Transformers [6] are deep learning models which adopt self-attention mechanism to process sequential input data. Pre-trained language models such as BERT [7] are pre-trained on extensive text data and provide enhanced capability on language data analysis, which is useful for content-based approaches to building recommender systems.

From Chapter 3 to Chapter 10, we will describe our approaches and experiments and we will have an sumarized discusion and conculsion on all of them in the Chapter 11 and 12.

3. STATISTIC ANALYSIS

We started from trying the simplest, intuitive Statistic Analysis. According to the course selected in the training data set, we counted the number of times each user feature co-occurs with each course. After that, we mapped the features of a target user to the number of its co-occurrences of each course. Then, we directly summed up those co-occurrences and recommended from the course with the highest number of co-occurrences to the lowest one.
For Seen Task, we additionally recorded the numbers of co-occurrences of arbitrary two courses. When predicting, similar with the Unseen Task, we summed up those co-occurrences and recommended from the course with the highest number of co-occurrences to the lowest one. Although this is a rough method, it can quickly give us a base line. It is useful for checking our models later. At least the result must be exceeded this base line that our models can be considered to have learned something. In addition, static analysis also gave us some insights about some unexpectedly important features.

4. DECISION TREE/RANDOM FOREST

In this method, we encoded the features of each user as a sparse feature vector and encoded the subgroups/courses 0 for not taken and 1 for taken as multi-output. Since the task requires to sorted by the probability, we choose random forest regressor instead of classifier. To reduce the training time, we use the xgboost library to run the random forest algorithm on GPU.

From previous work of a static analysis, we knew that the course a user would take has high correlation with the course he had taken. Hence, we randomly partitioned the taken course for each user into two part, one of which was taken as new features and the other was taken as training label, many times for training. For validation and testing, treat every taken course for each user in training set as features to predict the recommended courses.

5. LIGHTFM

Since the user features in the dataset are sparse, we can apply LightFM [4] on it, which is designed for the sparse data user provided. LightFM is a hybrid matrix factorization model which can deal with the cold-start problem and the sparse data. It describe every user with features f_u^U and describe a item with f_i^I. Then predict a user with the followings.

$$predict_{u,i} = Sigmoid(q_u \cdot p_i + b_u + bi)$$

Where $q_u = \sum_{j \in f_u} embedding_j^U$, $p_i = \sum_{j \in f_i} embedding_j^I$, and b_u, b_i for bias of user and item. Thus, it can help us deal with unseen users with their profiles. During the training process, we find it fits the training dataset fast accompanied by a high evaluation score 0.30. But, the result makes us upset that the score on Kaggle fall in a range between [0.10 0.20]. Which indicates that we overfit the dataset.

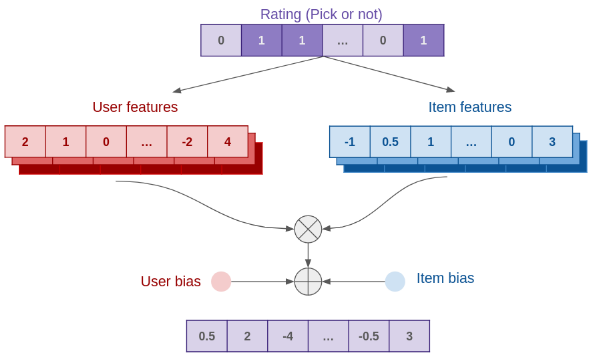

Fig. 1. LightFM

6. NEURAL FACTORIZATION MACHINES

Due to the high but sparse dimension of the user feature vector, NFM embedded the feature into a latent space and modeled the interactions between features via an inner product as in factorization machine(FM). Furthermore, to improve the linearity of FM, NFM used the non-linearity of neural networks in modeling higher-order feature interactions. The first layer of NFM was the embedding layer, which embedded the user features. The second layer of NFM was the bi-interaction layer, which is the dot product of each non-zero embedding.

$$f_{BI}(\mathcal{V}_x) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} x_i v_i \odot x_j v_j$$

where $\mathcal{V}_x = \{x_1 v_1, ..., x_n v_n\}$, x_i is non-zero feature and v_i is the embedding for i-th feature.

For computation simplicity, f_{BI} can be change to the following form.

$$f_{BI}(\mathcal{V}_x) = \frac{1}{2}((\sum_{i=1}^{n} x_i v_i)^2 - \sum_{i=1}^{n} (x_i v_i)^2)$$

For the deep neural network part, NFM would pass the output of the bi-interaction layer to a MLP as the following.

$$f(x) = \mathbf{h}^T \sigma_L(\mathbf{W}_L(...\sigma_1(\mathbf{W}_1 f_{BI}(\mathcal{V}_x) + \mathbf{b}_1)...) + \mathbf{b}_L)$$

where σ_L is the activation function and $\mathbf{W}_i$ and $\mathbf{b}_i$ was the weights and bias for each layer.

And for finally prediction will be

$$\hat{y}_{NFM}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + f(x)$$

where former part is a linear layer and the later part, $f(x)$, is the output of the MLP.

We take the top-k prediction task as a muti-lable task. Hence, we user the binary classification loss.

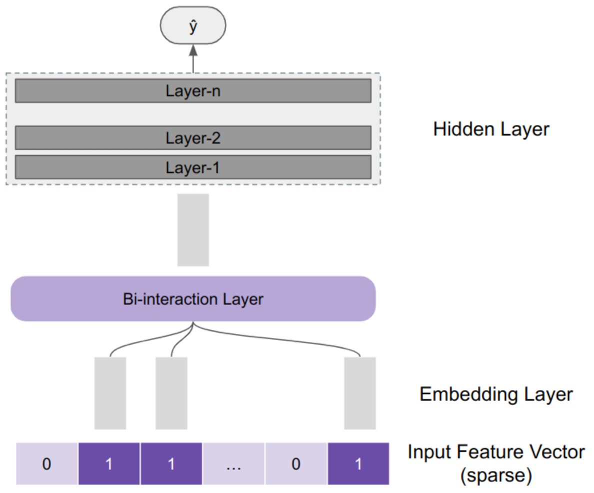

Fig. 2. Nerual Factorization Machine

7. MULTI-LAYER PERCEPTRON

In the multi-layer perceptron method, we encoded the features of each user as a sparse feature vector the same as the decision tree method. And encode the taken subgroups/courses the same as the decision tree method. For the model, we chose 2-layered MLPs and 3-layered MLPs for the recommendation for subgroup and the recommendation for course respectively. Since we treated the recommendation for each subgroup/course as a multi-label task, we chose binary cross entropy for each label.

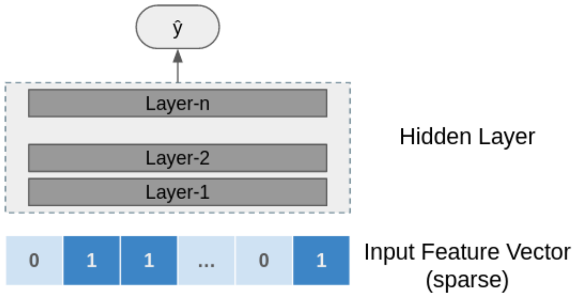

Fig. 3. Multi-layer Perceptron

The same as the experiment done in the decision tree method, we tried to treat the taken course as the features of a user. Hence, we randomly picked some taken courses as user's features and the other as labels while training. Then, we treated every taken course in the training set as the features of a user while predicting.

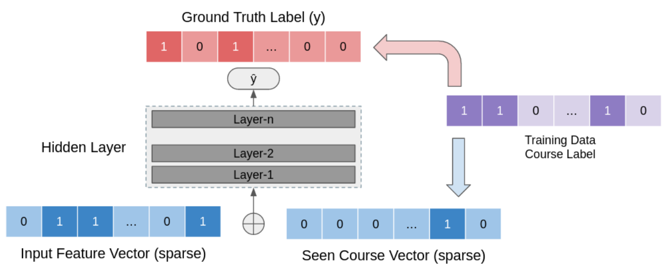

Fig. 4. MLP with Seen Course

In order to enrich the training data, we also try to randomly mask some users' features as a method of data augmentation while training.

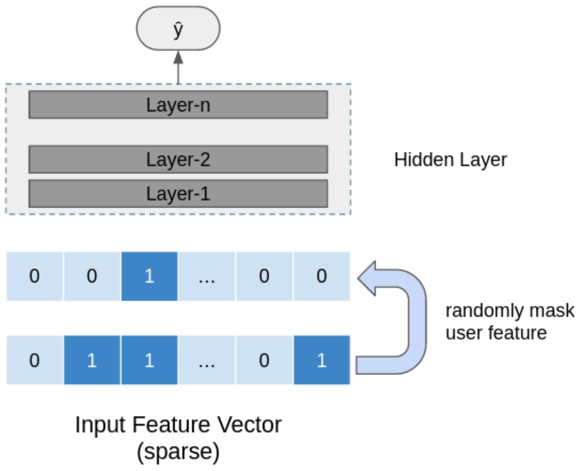

Fig. 5. MLP with Masking

8. PRE-TRAINED NLP MODELS WITH MLP

In this method, we make use of the word embeddings and the tokenizers of the Bert-family model. In this method, we just simply concatenate all the personal information to represent a person. Then project this sentence whose length is len_{seq} onto a len_{course} tensor with MLP, which we take as a score for each course. And since we only know the course user bought before, we can only use a binary vector to represent the target. Since the target is a binary vector, we took BCELoss as the loss function of the model. Thus, if we put a user as input, whoever is seen or not, the model will output an array with a length of courses. With the array, we can simply rank them by their output value.

To more utilize the course information, we also tried to concatenate the information after the user features and changed the task to a simpler yes-no question. In order to make course information readable by the model, we concatenate the different features of the courses in a specific pattern and turn course information into a readable sentence. After that, we put these sentences to a pre-trained BERT and get its embedded result. Finally, we concatenate the embedded course information with user features and feed it into an MLP model.

The performance of this method is not bad. However, it needs much more prediction, since it can only give the score of a course for a user at one time, which also results in a much longer training time. So we prefer to use cosine similarity of the user information and course information in the later trying, which is more efficient and can also consider the information of courses into the prediction.

9. PRE-TRAINED NLP MODELS WITH COSINE SIMILARITY

9.1. Fine-tune NLP Models

In this method, we assumed that the latent semantic space could help us to construct the relation between users and items. What's more, the descriptions of the users and the descriptions of the taken courses were close in the latent semantic space, which could be measured by the cosine similarity of their position in the latent space.

For data, we constructed the descriptions for users from their genders, occupation titles, recreation names, and interests at first. Secondly, we constructed the descriptions for courses from their names, groups, subgroups, and so on.

The model was mainly constructed by two pre-trained NLP models, one for user description and the other for course description.

$$enc_{user} = Bert\_Encoder_{user}(user\ description)$$

$$enc_{course} = Bert\_Encoer_{course}(course\ description)$$

For each pre-trained NLP model, there was a self-attention pooling[8] after it to pack the sequence output to only one vector for each user/course description.

$$x = Self\_Attention(enc)$$

$$enc = [e_0, e_1, e_2, .., e_n]$$

$$W = Softmax(Linear(enc)) = [w_0, w_1, ..w_n], w_i \in N$$

$$x = \sum_{i=0}^{n} w_i \times e_i$$

And these vectors, x_{user} and x_{course}, are exactly the position of the user and course description in the latent space. And then, we calculated the cosine similarity of the vector represented for the user description and the course description. Finally, map the value of cosine similarity from $[-1, 1]$ to $[0, 1]$ by sigmoid function as the probability to choose the subgroup/course. Since we treated the recommendation for each course as a multi-label task, we chose binary cross entropy for each label and calculate the gradient back to the pre-trained NLP models.

We update the pre-trained NLP models in previous part. We tried to fixed the parameters of the pre-trained NLP models. The only updating part of the models would be self-attention pooling layer in previous model architecture. Hence, we add a 2-layered MLP after self-attention pooling layer to add more model capacity. The loss function is the binary classification loss.

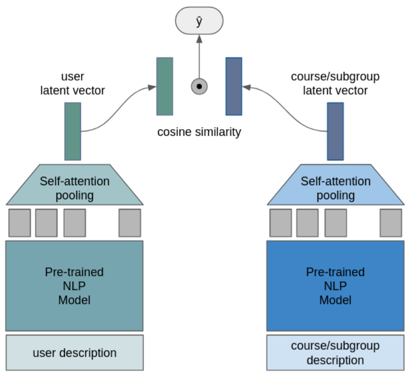

Fig. 6. Pre-trained NLP Models with Cosine Similarity

9.2. Multitask

We tried to share the NLP models and self-attention layer from both the user description extractor and the course description extractor. Hence, we replaced two pre-trained NLP models and their corresponding self-attention layers with a pre-trained NLP model and a single self-attention layer. We knew that it was necessary to update the pre-trained NLP models from the previous experiments, so we will update them in this method. After the description passing through the shared pre-trained NLP models and self-attention pooling layer, the output of the pooling layer would passes to an unshared 2-layered MLP respectively.

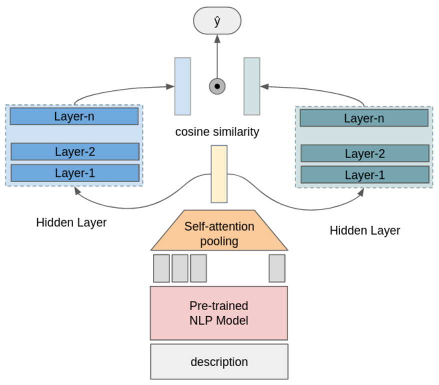

Fig. 7. Multi-task with Pre-trained NLP Models

10. TRANSFORMER ENCODER

10.1. Multi-layer perceptron

Inspired by NFM[5], we know that using embedding for each user feature to fix the problem of the user features that are high-dimension but sparse. We try to train the embedding and the model simultaneously. For initialization of our feature embedding, we random the value uniformly with the mean is 0 and the standard variance is 0.1. The backbone model we choose is the Multi-head Transformer Encoder[9]. And we used self-attention pooling layer[8] after it to pool the sequence of a encoded output to a vector. We view this output vector as the position of the user feature in the latent space. After the self-attention, we add the MLP as the last layer.

For a feature that the user is with, the input sequence will take the corresponding embedding. For the feature that the user is without, the input sequence will be filled with zero in the corresponding position and attention mask with be True as well to make the attention at this position to be infinitely small.

For loss function, we choose the binary classification loss as well. Since we view the recommendation task as a multi-lable task as previous methods.

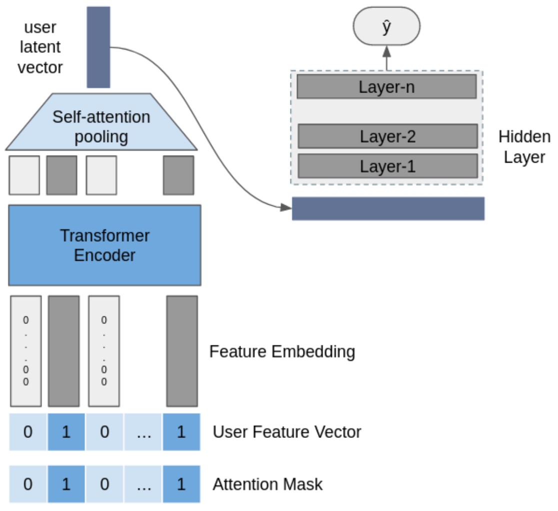

Fig. 8. Transformer Encoder with MLP

10.2. Cosine Similarity

From previous experiment on pre-trained NLP model with cosine similarity, we know that the cosine similarity can be used to measure the distance of two latent space vectors in this recommendation task.

In this method, we remove the final MLP layer. And we introduce another embedding for the subgroups or the courses. And calculate the cosine similarity of the output of the self-attention layer, which we view as the position of the user feature in the latent space, and the embedding of the subgroups or the courses. The loss function is the same as the previous method.

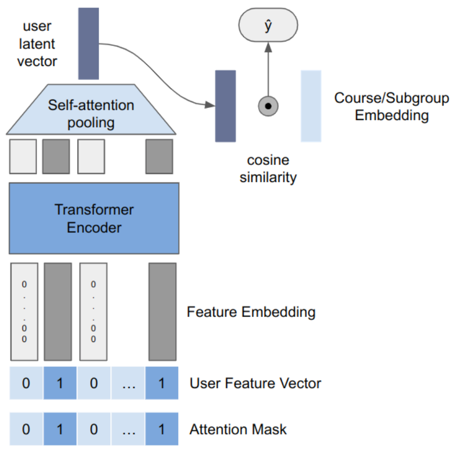

Fig. 9. Transformer Encoder with Cosine Similarity

10.3. Course Feature Extractor

Instead of purely calculating the cosine similarity from course embedding, we introduce a course feature extractor into the model to extract the course feature.

The course feature extractor is a structure similar to the user feature extractor. The course features we choose are groups, subgroups, and topics. The output of each course feature extractor will be concatenated to a unique course embedding for each course. And calculate the cosine similarity with the user latent vector and the concatenated course latent vector. The loss function is the same as the previous methods, which is the binary classification loss.

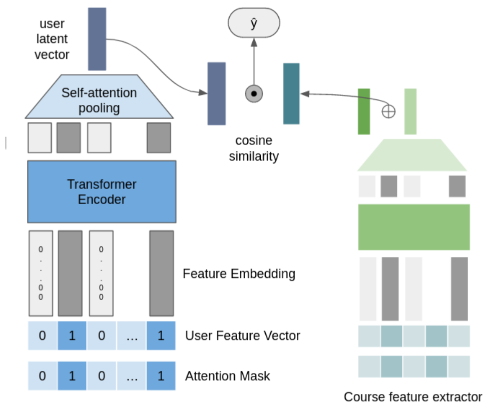

Fig. 10. Transformer Encoder with Course Feature Extractor

11. DISCUSSION

Below are the scores of each models base on validation dataset.

Methods	Seen Course	Unseen Course
Statistic Analysis w/ User Feature	0.06546	0.07594
Statistic Analysis w/ Seen Subgroup	0.08101	N/A
Statistic Analysis w/ Seen Course	**0.08933**	N/A
Random Forest	0.07064	0.10401
Random Forest	0.07926	N/A
Sparse Feature Array w/ MLP	0.07858	0.12140
Sparse Feature Array w/ MLP w/ seen course	0.08525	N/A
Pre-trained model w/ MLP	N/A	0.07088
NFM	0.07771	0.11946
Transformer Enc w/ MLP	0.07802	0.12255
Transformer Enc w/ MLP w/ seen course	0.08888	N/A
Transformer Enc + Cosine Similarity	0.07508	0.12484
Transformer Enc + Cosine Similarity w/ seen course	0.08528	N/A
Transformer Enc + Cosine Similarity w/ feature extractor	N/A	**0.12607**

Table 1. Validation score on course prediction task

We found that in the task of the ”Seen User Course”, the relationship between courses is not the most significant factor. We found that the user’s purchase history and the popularity of the course have a greater impact on the results. Therefore, the models we used that are more relevant to user embeddings and course embeddings did not perform very well in this task, and even the results on the validation data were worse than the results of the statistical analysis. In the ”Seen User Course” task, we can observe that the performance of conventional statistical analysis and traditional machine learning methods such as random forest is not bad.

However, in the task of the ”Unseen User Course”, because there is no purchase record of the user, the prediction requires a model to learn the audience of each course and the meaning each course represents. It is very suitable for the Transformer encoder we tried later to give the embedding of the course, therefore, the performance of our model on this task has significantly surpassed the results of statistics methods.

Methods	Seen Topic	Unseen Topic
Statistic Analysis w/ User Feature	0.23346	0.19140
Statistic Analysis w/ Seen Subgroup	0.25794	N/A
Statistic Analysis w/ Seen Course	0.25659	N/A
Random Forest	0.27391	0.28779
LightFM	N/A	0.30343
Sparse Feature Array w/ MLP	0.28368	0.31957
Sparse Feature Array w/ MLP + mask	0.28378	0.29874
NFM	0.28300	0.31869
Transformer Enc w/ MLP	0.28199	0.32014
Transformer Enc + Cosine Similarity	**0.28454**	**0.32785**

Table 2. Validation score on topic prediction task

We think that, comparing with, the topic selected by a user will actually be closer to a user’s interest, so we will need more meaning represented by a topic in prediction. So this allows our Transformer encoder to give users data and course embeddings a more significant effect than pure data analysis. We also found that NFM, a non-linearity model, performs better than the linearity model LightFM. The result also reflects on the MLP one and LightFM. We believe that the relation between topics can not be represented by a simple linear relation. Thus, a model with activational layer often performs better.

12. CONCLUSION

We applied some models in recommendation algorithm to solve the top-k recommendation problems in this final project, like LightFM, random forest and multi-layer perceptron. We proposed a method that combine transformer encoder with cosine similarity, which beats a common recommendation system model LightFM in our experiments. In addition, we also found some key points in the process: At the beginning, we paid more attention to the course information, such as how to make better use of the descriptions of all courses, the information introduced by chapters, or analyze the relationship between courses. So we also tried to use some NLP models to try to train the model to understand the semantics of the course at the beginning. Although this type of model can be trained with enough resources to produce good results, it is a bit too time-consuming compared to our later method. And we later discovered in the seen task that the user's course purchase record and the popularity of the course have a greater impact on the results. So we changed to train course and user embedding through the user's purchase records and got pretty good results. On the seen course task where our performance is relatively poor, we can consider strengthening the analysis of the user's purchase records and the popularity analysis of the course, which may lead to better performance.

13. WORK DISTRIBUTION

學號	姓名	contribution
B09902016	鮑鈺文	NFM, Random Forest, MLP, Transformer Encoder (MLP, cosine similarity), Pre-trained NLP model (cosine similarity)
B09902017	李安傑	Statistic Analysis, Pre-trained NLP model w/ MLP and Binary Classification, Transformer Encoder (course feature extractor)
B09902019	沈立程	Pre-trained NLP model w/ Multi-Label Classification
B09902135	賴豊彰	LightFM, Pre-trained NLP model w/ MLP

Table 3. Work Distribution

14. REFERENCES

[1] Donghui Wang, Yanchun Liang, Dong Xu, Xiaoyue Feng, and Renchu Guan, "A content-based recommender system for computer science publications," *Knowledge-Based Systems*, vol. 157, pp. 1–9, 2018.

[2] John Breese, David Heckerman, and Carl Kadie, "Empirical analysis of predictive algorithm for collaborative filtering," *UAI*, 01 2013.

[3] Tin Kam Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, 1995, vol. 1, pp. 278–282 vol.1.

[4] Maciej Kula, "Metadata embeddings for user and item cold-start recommendations," in *Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 16-20, 2015.*, Toine Bogers and Marijn Koolen, Eds. 2015, vol. 1448 of *CEUR Workshop Proceedings*, pp. 14–21, CEUR-WS.org.

[5] Xiangnan He and Tat-Seng Chua, "Neural factorization machines for sparse predictive analytics," in *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, 2017, pp. 355–364.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *ArXiv*, vol. abs/1810.04805, 2019.

[8] Pooyan Safari, Miquel India, and Javier Hernando, "Self-attention encoding and pooling for speaker recognition," 2020.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention is all you need," 2017.