GEM-D: Graph Embedding approach on MOOC's early Dropout prediction

Antonio Campanozzi[1]

[1]University of studies Aldo Moro, Bari, Apulia, Italy

Abstract

Since the COVID-19 pandemic, Massive Open Online Courses (MOOCs) have gained significant popularity. However, due to their fully online nature, they suffer from a notoriously high dropout rate. Numerous Machine Learning and Deep Learning approaches have been proposed to address this issue. Among the most critical challenges is the lack of earliness in the prediction, needing the gather of substantial amount of information before being able decide whether the dropout is likely or not.

In this report, I present a novel approach to this problem by leveraging graph embeddings to generate synthetic yet plausible user-course interactions. These enriched representations are then used to predict student dropout at an early stage, potentially providing actionable insights into learner engagement.

Keywords

Graph Embedding models, Relational learning, Dropout prediction, Binary classification,

1. Introduction and Motivations

The primary focus of this project is to explore an alternative method for the early prediction of course dropout. In many contexts, it is assumed that a substantial number of user interactions with a product (in this case, an online course) are recorded and that these interactions are sufficient to be fed into a trained model for evaluation. However, in real-world scenarios, users often lack explicit interactions, or the recorded interactions carry limited predictive value.

For this reason, I investigated the effectiveness of a link prediction approach based on graph embeddings, leveraging the structural properties of the graph constructed from existing interactions between users and courses.

Link prediction is employed as a tool to generate feature vectors representing actions between a new user-course relationship. These vectors are then input to a binary classifier that predicts whether a user will drop out of a course based on their observed actions.

This approach is designed to provide both plausible samples of user behavior across multiple courses and an early prediction outcome, that can potentially be integrated within a recommendation system, to prioritize suggesting courses that users are less likely to abandon.

The remainder of this report explores the various components of the proposed solution in detail. Section 2 provides a brief overview of related work in the research that inspired this methodology. Section 3 outlines the development of the proposed architecture. Section 4 presents the evaluation methodology and the obtained results. Finally, Section 5 concludes the report by discussing current limitations and proposing potential directions for future work.

2. Related Work

The final stage of the pipeline can achieve excellent results even when utilizing simple Machine Learning models, without relying on complex neural architectures. In *Dropout Prediction in MOOCs Using Deep Learning and Machine Learning* [1], various models from both machine learning and deep learning domains were evaluated on well-known MOOCs datasets for binary classification to predict whether

Machine Learning Course 2024-2025

a.campanozz@studenti.uniba.it (A. Campanozzi)

a user would drop out of a course. The results demonstrate strong performance of machine learning models, particularly non-linear ones and those based on ensemble learning techniques.

The studies [2, 3] address the need for early dropout prediction by employing machine learning models whose parameters have been optimized to better fit the problem. These approaches still require at least one week's worth of user-course interaction data to generate reliable predictions.

SIG-Net [4] is the closest reference work to the approach proposed here, as it introduces a graph-based method for dropout prediction, specifically using Graph Neural Networks (GNNs). This study highlights that a major limitation of pure machine learning methods lies in the difficult feature extraction process and their neglect of the rich information contained in user-course interactions, which tends to be flattened in such models. SIG-Net generates embeddings from a user-course interaction graph and feeds these embeddings into a Bi-LSTM for classification.

3. Proposed Approach

The proposed work is structured as the following pipeline:

- Extraction of triples from the raw dataset
- Hyperparameter optimization, training, and validation of graph embedding models
- Dataset processing for dropout prediction, encoding users and courses using the best-performing graph embedding model
- Hyperparameter tuning, training, and validation of the classifier
- Inference using the selected graph embedding model to obtain the highest-scoring relationships for a given user-course pair
- Generation of the feature vector
- Validation of the generated sample using the trained classifier

All action on Graph embedding models were performed using `pykeen`, a Python library based on `pytorch` for training and evaluating Knowledge Graph embeddings.

3.1. Description of the Datasets

The reference dataset tracks a wide range of user interactions with MOOCs on the Xuetang online platform, developed by Tsinghua University in China. It includes the following features:

- **`username`**: A unique identifier for the student who performed the actions.
- **`course_id`**: A string uniquely identifying the course.
- **`session_id`**: An alphanumeric identifier for a session grouping a sequence of user actions during a single platform interaction.
- **`action`**: A string describing the specific user action, such as `load_video`, `play_video`, `pause_video`, etc.
- **`time`**: A timestamp recording when the event occurred.
- **`truth`**: A binary label indicating whether the user dropped out of the course.

Given the dataset's large size (6.5 GB), only the test split (1.7 GB) was used for this study. This subset still offers a rich and computationally manageable dataset for experimentation, while remaining representative of the overall task.

Two different preprocessing operations were applied: the first to extract triples for the graph embeddings, and the second to create a suitable format for training the classifier.

Triples were defined as `<user, action, course>`. To do so, I simply extracted from the dataset the three columns regarding the elements of the triples, eliminating duplicates. A total of approximately 500,000 triples were generated, describing the interactions of 44,000 users with 200 courses, involving 20 distinct relations.

Due to limited computational resources and lack of expertise in temporal link prediction, only the static nature of these interactions, without information of the time of execution of the action, was considered. As a result, timestamp and session-related attributes were also excluded from the classification dataset. Instead, a simplified 0-1 encoded matrix was generated, where a "1" indicates that a specific user performed a given action on a course, and "0" otherwise. User and course IDs were mapped using the `TriplesFactory` module provided by `pykeen` to ensure consistency with the embedding space.

user	course	action_1	action_2	action_3	...	truth
10102	44040	1	0	0	1	1
10102	42984	1	1	0	0	1
19829	42984	0	1	1	0	0
16789	42984	0	0	1	0	1

Table 1
Classification dataset structure.

The resulting dataset consists of approximately 67,000 samples, forming a solid basis for model training.

3.2. Models Leveraged

The main focus of this study is on graph embedding models. For a more comprehensive analysis, three representative models were selected:

- **TransE** [5]: The simplest among the three, TransE is a distance-based approach where relations are modeled as translations in the embedding space. The scoring function evaluates the proximity between the head plus the translated relation to tail entities.
  This model was chosen based on the principle of Occam's Razor: its simplicity, if combined with strong results, can make it a competitive candidate.
- **RotatE** [6]: A more expressive alternative, capable of modeling relational properties such as symmetry, antisymmetry, composition, and inversion. It embeds entities and relations in a complex vector space, where each relation corresponds to a rotation from the head to the tail. The scoring function builds upon the same principle as TransE.
- **TuckER** [7]: The most complex of the three, TuckER is a similarity-based model that applies Tucker decomposition to the binary tensor representing the knowledge graph triples. It was selected for its conceptual divergence from the other two models, its reasonable computational cost for a complex model, and the promising results reported in the original paper.

For the classification task, **XGBoost** was selected. As demonstrated in [1], this model achieves state-of-the-art performance on the Xuetang MOOC dataset, also used in this project.

All models underwent extensive hyperparameter tuning to ensure optimal adaptation to the data while minimizing overfitting and maximizing generalization.

3.3. Hyperparameter Tuning

Hyperparameter tuning played a crucial role in achieving strong results. The large number of variables involved, both in the training procedure and model definition, makes it unrealistic to rely on default settings. With real-world data, the presence of noise justifies several preliminary attempts before identifying the optimal configuration for the sample space.

All models in this project were optimized using `optuna`, a widely adopted hyperparameter optimization framework. PyKEEN provides an `hpo` module based on Optuna, which allows for simple configuration of the optimization through the `hpo_pipeline` method.

For graph embedding models, it is useful to distinguish between hyperparameters related to the model architecture itself and those that directly affect the training process.

Model-specific hyperparameters:

- **`model.embedding_dim`**: Defines the dimensionality of embedding vectors. Higher dimensions increase the chance of capturing more information but also raise computational costs. values from 100 to 400 were utilized
- **`model.entity_initializer`**: Randomly initializes the entity embeddings before they are refined during training.
- **`model.dropout_0,1,2`**: Present in TuckER and used as regularizers. As suggested in [7], for datasets with a high number of triples per relation (such as the one used here and WN18RR), lower dropout values (0.1–0.3) are recommended. Accordingly, only this range was considered.

Training-related hyperparameters:

- **Optimizer**: A very low learning rate (10^{-5} to 10^{-2}) was selected to ensure stable, less aggressive updates.
- **Regularization**: Evaluated only for TransE and RotatE (TuckER already uses dropout). A conservative regularization magnitude was chosen to avoid premature convergence. Both L1 and L2 penalties were separately considered.
- **Negative sampling**: For each positive triple, a fixed number of negatives can be generated to enrich the training signal. While more negatives may improve model robustness, they significantly increase computational load. Given available resources, the number of negatives was capped at 14 per sample.
- **Loss function**: For TransE and RotatE, the *Margin Loss* was used. The lower the margin, the less severe the loss score.

The main objective was to find a configuration that yielded promising results already on fewer epochs, but without converging too quickly. In fact, these models typically require many epochs to produce high-quality embeddings. For all three graph embedding models, 25 trials of 50 epochs each were conducted, aiming to maximize ranking-based metrics.

XGBoost Tuning

XGBoost was tuned across 70 trials with the goal of maximizing the AUC score, computed as the average one across 5-fold cross-validation on each trial configuration.

Among the various hyperparameters, the most influential ones are listed below:

- **`learning_rate`** $\in [10^{-4}, 0.3]$ (log scale): Controls the step size at each boosting iteration. Lower values generally lead to better generalization but require more trees to converge.
- **`max_depth`** $\in [3, 15]$: Sets the maximum depth of the trees. Shallower trees help reduce overfitting; deeper trees increase model capacity.
- **`gamma`** $\in [0.1, 10]$ (log scale): Minimum loss reduction required to perform a split. Higher values act as stronger regularization.
- **`subsample`** $\in [0.5, 1.0]$: Proportion of samples used per boosting round. Helps reduce overfitting and adds randomness.
- **`reg_alpha` and `reg_lambda`** $\in [1, 10]$ (log scale): L1 and L2 regularization terms respectively, used to penalize overly complex models.
- **`n_estimators`** $\in [500, 2500]$: Total number of trees. A higher upper limit ensures convergence even when the learning rate is small.

3.4. Sample Generation

The best-performing graph embedding model was employed for sample generation. The strategy adopted is relatively simple and proceeds as follows:

1. Given a user ID, the model predicts the most likely course for each type of relation.
2. For each predicted course, the user-course pair is fixed, and all possible relations are scored by the model.
3. Only the top-k predicted relations are considered as completed and are assigned a value of 1 in the resulting feature vector. The value of k is set to the average number of actions performed by users per course. This choice prevents the generation of feature vectors that are either too sparse or too dense, thereby reducing the likelihood of introducing outlier samples into the dataset.

3.5. Experimental Setting

All experiments were made on the P100 GPU offered on the *Kaggle*'s cloud platform. Each Graph embedding model was trained on 200 epochs with `batch_size` of 512, and an early stopping criteria with patience equal to 3, called every 25 epochs. Trained models, results, code and datasets are available at https://github.com/AntonioCampanozzi/Machine-Learning-24-25

4. Evaluation and Results

Graph embedding models were evaluated on the test triples using standard rank-based metrics:

- **Mean Rank (MR)**: the average position of the correct triple across all predictions. This metric is sensitive to outliers, which can significantly skew the mean.
- **Mean Reciprocal Rank (MRR)**: ranges in [0, 1]; it is the average of the reciprocal ranks of the correct triples. Higher values indicate better ranking quality.
- **Hits@k**: the percentage of times the correct triple appears in the top-k positions of the ranking. In this study, Hits@1, Hits@3, Hits@5, and Hits@10 were computed.

Model	Hits@1	Hits@3	Hits@5	Hits@10	MRR	Mean Rank
TransE	0.496	0.580	0.608	0.641	0.548	2050
RotatE	0.474	0.553	0.585	0.638	0.534	177.4
TuckER	**0.567**	**0.674**	**0.720**	**0.785**	**0.643**	**100**

Table 2
Graph embedding models results.

Although TransE initially appears comparable or even slightly superior to RotatE in some metrics (e.g., MRR), its extremely high Mean Rank indicates a lack of consistency. In several cases, TransE fails to rank the correct triple within a reasonable range, leading to performance degradation overall. RotatE, by contrast, shows greater stability with a much lower Mean Rank.

TuckER clearly outperforms both TransE and RotatE across all metrics. In particular, it achieves notable improvements in Hits@5 and Hits@10, suggesting it is especially effective at capturing meaningful relationships in the embedding space. These results, based on the findings in [5, 6, 7], and—given the real-world, noisy nature of the dataset—are considered competitive.

The XGBoost model was evaluated on standard binary classification metrics:

- **AUC (Area Under the ROC Curve)**: measures the model's ability to distinguish between classes across all classification thresholds. A higher AUC reflects better overall performance.
- **Precision**: the proportion of true positives among all predicted positives.
- **Recall**: the proportion of true positives among all actual positives.
- **F1-score**: the harmonic mean of Precision and Recall, offering a balance between the two.

Model	XGBoost
AUC	0.834
Precision	0.843
Recall	0.941
F1-score	0.890

Table 3
XGBoost classification results.

The performance achieved is in line with the results reported in [1], further supporting the effectiveness of this model. In particular, the high recall suggests that the model is very sensitive in identifying students at risk of dropout, which is critical in early intervention scenarios.

5. Conclusion and Limitations

Graph embedding models demonstrated a reasonable ability to project MOOC data into continuous space, while the XGBoost classifier maintained consistently the performance in dropout classification tasks. Nonetheless, several limitations and areas for future improvement have been identified:

- **Lack of temporal information**: As discussed, only static triples were considered in this study. Incorporating user-course interactions across multiple timestamps could capture richer behavioral dynamics, potentially leading to more informative embeddings and improved sample generation, still requiring a significative number of interaction sessions.
- **Computational constraints**: The GPU resources available limited the extent of hyperparameter tuning and the number of training epochs. With access to more powerful hardware, it would be possible to explore larger embedding dimensions, sample more negative examples per positive, and run longer training cycles to achieve more refined representations.
- **Sample generation strategy**: The current approach to generating classification samples is relatively basic. Exploring more advanced sampling strategies could lead to more representative and challenging artificial examples. This stage of the pipeline clearly deserves further experimentation and refinement.

References

[1] R. Basnet, C. Johnson, T. Doleck, Dropout prediction in moocs using deep learning and machine learning, Education and Information Technologies 27 (2022) 1–15. doi:10.1007/s10639-022-11068-7.

[2] T. Panagiotakopoulos, S. Kotsiantis, G. Kostopoulos, O. Iatrellis, A. Kameas, Early dropout prediction in moocs through supervised learning and hyperparameter optimization, Electronics 10 (2021) 1701. doi:10.3390/electronics10141701.

[3] A. Alamri, M. Alshehri, A. Cristea, F. Pereira, E. Teixeira de Oliveira, L. Shi, C. Stewart, Predicting MOOCs Dropout Using Only Two Easily Obtainable Features from the First Week's Activities, 2019, pp. 163–173. doi:10.1007/978-3-030-22244-4_20.

[4] D. Roh, D. Han, D. Kim, K. Han, M. Y. Yi, Sig-net: Gnn based dropout prediction in moocs using student interaction graph, in: Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, SAC '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 29–37. URL: https://doi.org/10.1145/3605098.3636002. doi:10.1145/3605098.3636002.

[5] A. Bordes, N. Usunier, A. Garcia-Durán, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, NIPS'13, Curran Associates Inc., Red Hook, NY, USA, 2013, p. 2787–2795.

[6] Z. Sun, Z.-H. Deng, J.-Y. Nie, J. Tang, Rotate: Knowledge graph embedding by relational rotation in complex space, 2019. URL: https://arxiv.org/abs/1902.10197. arXiv:1902.10197.

[7] I. Balazevic, C. Allen, T. Hospedales, Tucker: Tensor factorization for knowledge graph completion, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Association for Computational Linguistics, 2019. URL: http://dx.doi.org/10.18653/v1/D19-1522. doi:10.18653/v1/d19-1522.