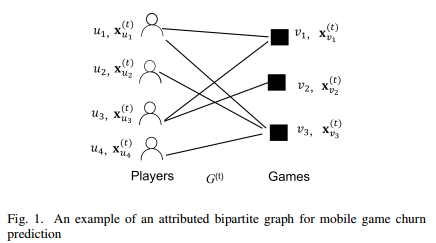
# Semi-supervised and inductive embedding model for churn prediction of large-scale mobile games

1. Introduction

* Previous studies on mobile game churn prediction using traditional ML models (e.g., logistic regression, random forests, Cox regression)
* Limitations:
* Developed for predicting churn of one or a few mobile games -> none is capable of handling the churn prediction of large-scale mobile apps and users
* User-app interaction data often comes with rich contextual info (e.g., wifi connection status, screen brightness, audio volume) >< not considered in previous studies
* Rely on handcrafted features -> usually cannot scale well in practice
* Contributions:
* First solution for **churn prediction of large-scale mobile games** using hundreds of millions of user-app interaction records
* Novel **semi-supervised and inductive model based on embedding** – capture the dynamics between users and mobile games based on introduced temporal loss in the formulated objective function. Model is able to **embed new users or games not used in training**
* Attributed **random walk technique** -> **sample contexts of edges** in an attributed bipartite graph that takes into account both topological adjacency and attribute similarities
* Comprehensive **experimental evaluation with large-scale real-world data** collected from Samsung Game Launcher

1. **Problem formulation**

* Churn: player stopping using a game within a given period (i.e., no app usage in the period). Duration T of the period may vary (e.g., T= 14 or T=30)
* In this paper, we consider the **generic game churn prediction problem without assuming any particular value of T**
* Note: uninstall is different from churn (there may be a large time gap between cessation of playing and uninstall, if any)
* Relationship between players and games – represented by an attributed bipartite graph



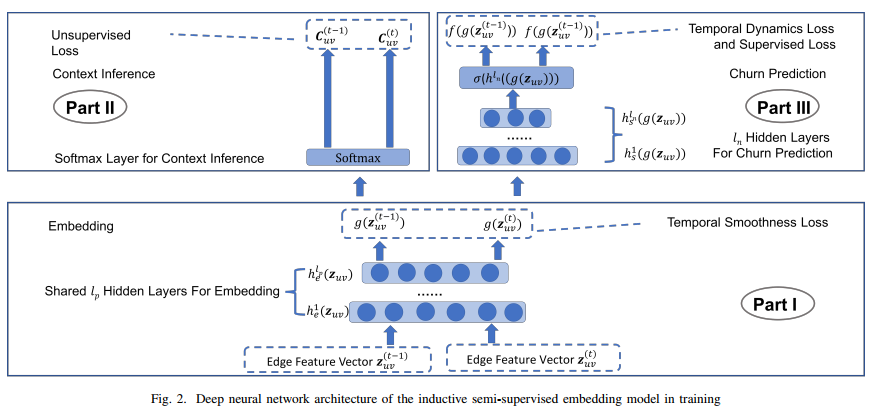
* **Notations**

|  |  |
| --- | --- |
| **Notations** | **Descriptions or Definitions** |
|  | Attributed graph at time |
|  | Attributed graph at time |
|  | Historical attributed graphs |
|  | Set of all player nodes in |
|  | Set of all game nodes in |
|  | Set of all edges in |
|  | Indicator of the existence of edge () in |
|  | Feature vector of user |
|  | Feature vector of game |
|  | Aggregated feature vector of edge |
|  | Number of attributes in |
|  | Number of attributes in |
|  | Embedding dimension |
|  | The edge embedding function |
|  | The churn prediction function |
|  | Number of prediction hidden layers |
|  | Number of embedding hidden layers |

* Let be the attributed bipartite graph at time
* Vertex set denote the set of users
* Vertex set denote the set of games
* Player represented by a node
* Game represented by a node
* Each user is associated with a feature vector , where is the size of
* Each game is associated with a feature vector , where is the size of
* There is an edge between nodes if player has played in the time window
* Set of edges denoted by
* Definition 1: Mobile game churn prediction
* Consider **collection of attributed bipartite graphs observed from time** , which is denoted by
* Let be the **indicator of the existence of edge** in
* **For any edge** , predict the prob. , that is the **probability that**

1. **Methods**
2. **Overview of solution**

* Recent progress in DL and graph embedding, a natural promising direction is to adopt graph embedding frameworks for churn prediction
* **Key technical challenges:**
* All existing methods are transductive -> cannot produce embeddings for new player-game pairs
* Existing methods are either purely supervised or unsupervised -> do not take full advantage of relevance between embedding and a task
* Existing methods are node-centric -> not directly applicable to edge related tasks
* Existing methods only handle a static graph and do not incorporate graph dynamics in embedding >< new players, new games, new player-game relationships every day
* **Propose: a novel inductive semi-supervised embedding model** in dynamic graphs that jointly learns the **prediction function f** and the **embedding function g**



* **DNN consists of 3 parts:**
* **Part I: produce embedding feature vectors** from raw edge feature vectors , where is the size of raw edge feature vectors
  + To learn the prob. of churn, we need to construct a feature vector for each
  + It is impractical to calculate features for all possible edges which may appear in the prediction period because the number of possible edges is large, which is
  + Instead, we **construct the feature vector** **from attribute-wise cosine similarity aggregation of similarity aggregation** of
* **Part II: infer contexts from embedding feature vectors**
  + Context of an edge – the edges that are **similar to and co-occur with the edge under some graph sampling strategy** (e.g., random walk)
  + Part I and II – unsupervised component of the model. Jointly trained by minimizing the **error due to incorrect context inference and inconsistency with temporal smoothness** (III-C for explanation)
  + Part I and II – trained in an inductive and edge-centric way – learn an **embedding function that generalizes to any unseen edges** as long as their feature vector is available
    - Novel attributed random walk to sample similar edges as contexts (III-D)
* **Part III: supervised churn prediction task from embedding feature vectors**
  + Trained by minimizing the error of incorrect churn predictions
* Supervised and unsupervised component are simultaneously trained in **a single objective function**
* Part III and Part II share the **common hidden layers** in part I -> latently coupled with each other -> helps embedding align with the supervised prediction task
* Part I and III both consider **graph dynamics** in training
* Part I: handle graph dynamics by requiring the **embeddings of the same edge at 2 consecutive timestamps to stay close**
* Part III: handles graph dynamics by requiring the **churn prob of the same edge at 2 consecutive timestamps to follow a decaying pattern**
* **Objective function composed of 4 parts:**

(1)

Where:

1. Static Loss Functions

* **k-th hidden layer for churn prediction** (prediction hidden layer)
* Model churn prediction by  **hidden layers**
* **Prediction output layer** represented by **(2)**:

Where:

* **Supervised loss** (3):
* Part II: **embed handcrafted features** **into a latent space**  where m is the size of latent space
* **K-th hidden layer for embedding** (embedding hidden layer)
* **layers in part I to represent embedding function g**
* **Embedding output layer**: (4)
* **Unsupervised loss function**:

Where:

* + . Contextual edges are obtained by attributed random walk on the bipartite graph (discussed later)
* Likelihood of having a contextual edge conditional on the embedding of

**(6)**

* Weights for regularization consists of
* Regularization part can be expressed as:

(7)

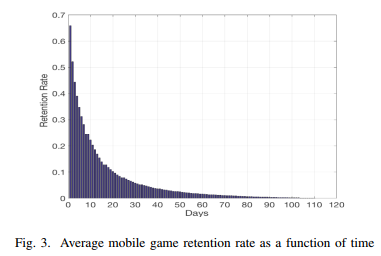
Where are trade-off weights on different regularization terms

1. **Temporal Loss Function**

* Loss related to graph dynamics
* **Observation 1**: For a given user-game play relationship, **the longer the relationship exists, the more likely the user is to churn the game**
* 71% of all mobile app users churn within 90 days (higher for mobile games)
* Average retention rate of mobile games as a function of time: 95% of user-game play relationships end after 40 days

(8)

Where



* **Observation 2:** For a given **user-play relationship** denoted by an edge in the attributed bipartite graph, its **context usually evolves slowly at 2 consecutive timestamps**
* Topology and attribute values of the attributed bipartite graph mostly evolve smoothly at 2 consecutive timestamps -> similar contexts for a given edge at consecutive timestamps
  + - Embeddings at these 2 timestamps should also be close – temporal smoothness

(9)

* **Observation 3**: by definition, churn in nature introduces **right censoring** to the training dataset
* Observation period – some time duration in history from time
* Data for train and test all come from the observation period
* Label of a player-game pair at a specific timestamp is determined by their interaction in the next T time duration -> labels of some player-game pairs in the last T duration could be unknown
* Introduce a binary indicator to indicate whether an edge (u,v) is censored at timestamp i (0: censored, 1: otherwise)
* Inequality in (8) is updated by:

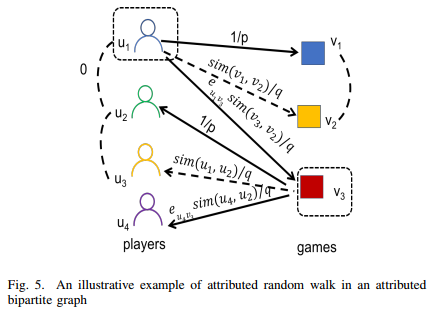
(10)

Where denotes the timestamp when the edge (u,v) was observed

* After timestamp , the label of (u, v) becomes unknown
  + - More reasonable to require that (10) holds pairwise between time point
* **Temporal loss (11)**:
* First term = temporal smoothness
* Second term = temporal dynamics
* Taking (2) and (4) into (11):

1. Context generation by attributed random walk

* Goal: embed an edge (not a node)
* A simple topology-based random walk may return 2 adjacent edges having the same player or the same game while totally ignoring thee similarity of the other end
  + - Undesirable
* Attributed random walk measures similarity by attributes and allows to transit to similar nodes even if they are not connected
* Novel attributed random walk technique that takes into account both topological adjacency and attribute similarities to make the transition decision of the walk
* Example of attributed random walk in an attributed bipartite graph



* Solid line – there exists an edge in the attributed bipartite graph
* Denote type of node o by type(o) – Node type can be of value either player or game
* Dashed directed lines – do not exist in the original attribute graph but may be considered as transitions by attributed random walk due to attribute similarities
* Dashed undirected lines – added augmented edges between nodes and their similar same type nodes
* Time-consuming to calculate pairwise similarities between a node and all other nodes with the same type
* Add augmented edges for a proportion of the same type nodes
* Augmented edges of node is a filtering param and:
* Consider a random walker that just traversed edge and now resides at node
  + - Decide which node to transit to
* Attribute similarities matter -> cannot just evaluate those nodes that are neighbors of
* Need to evaluate all nodes of the same type within the 2-hop neighborhood of
* One-hop same type adjacent nodes of :
* Two-hop same-typed neighbor nodes of :
* Transition prob in attributed random walk can be calculated as follows:

Where p and q are normalization constants used to control the walk strategy

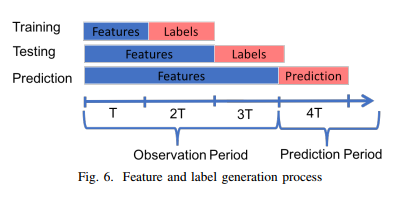
* The attributed walk in an attributed bipartite graph walks through different types of nodes repeatedly
* Enlarges the prob that any 2 consecutive edges in a path are similar and thus the prob that the whole set of edges on the attributed random walk path are similar
* When choosing next node to visit:
* Nonzero prob to choose those that are not connected to the present node but share similar attributes to the previous node
* For those connected to the present node, the prob to choose from them is weighted by the similarity between the descendant and the ancestor nodes
  + - An edge can be the context of another even when they are not adjacent but similar in both ends

1. Experimental evaluation

* Compare the semi-supervised (SS) model with:
* LR – logistic regression based solution
* RS – supervised variant of the SS mode (loss function only contains the supervised component and the regularization term)
* DT – decision tree
* RF – random forests
* SVM – support vector machine
* In the experiments, consider the churn duration T = 14
* Proposed solution is not restricted to any particular choice of T

1. Dataset and feature/label construction

* Features and labels – taken from disjoint periods to avoid data leakage



* Labels: whether a player churns a game on that day
* Features: constructed from historical data before the day to be predicted
* Training and testing set are split by label days in chronological order
* Taking labeled data in the first 2/3 of the observation period as training set and the remaining 1/3 as testing set -> ensure there is a time difference between testing and training set

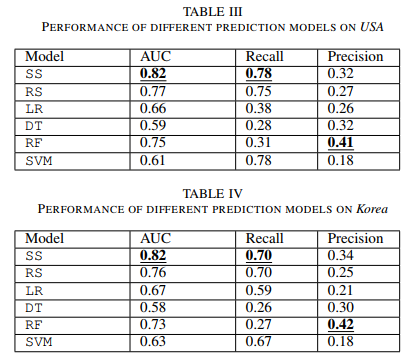
1. Experimental settings

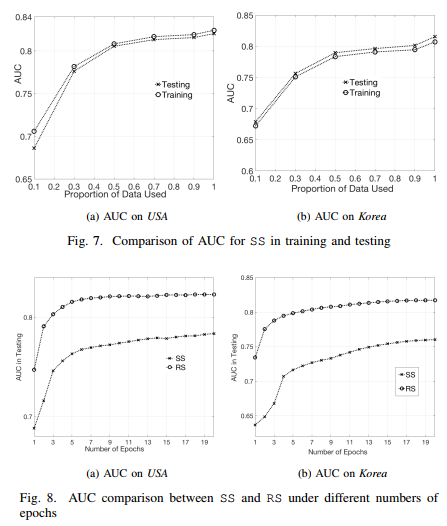
* Hyperparams tuned: grid search and combination yielding the best performance is chosen
* Learning rate
* Batch size
* Regularization terms
* Number of players
* Number of neurons per player
* Regularization params – all set to 1
* in (1) set to be 0.02, 0.01, 1e-5
* – set to be 1, 1, and 0.05
* Parameters used for training the deep neural network:

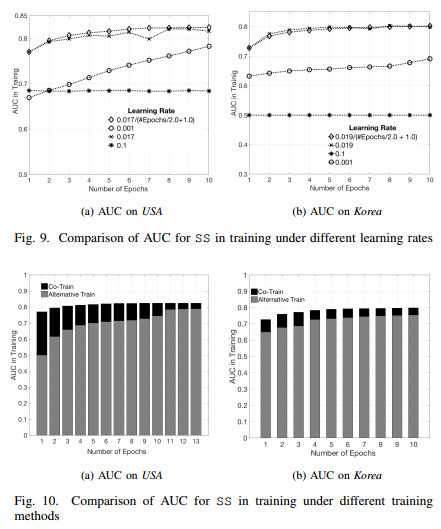
|  |  |  |
| --- | --- | --- |
| Param | USA dataset | Korea dataset |
| Player feature dimension | 10042 | 10042 |
| Game feature dimension | 10042 | 10042 |
| Player-game feature dimension | 30 | 30 |
| Learning rate | Initial value = 0.017  Decay by where k is the number of epochs | Initial value = 0.019  Decay by where k is the number of epochs |
| Number of neurons | Input layers 30  Embedding layers 50  Output layers 380k | Input layers 30  Embedding layers 50  Output layers 632k |
| Number of epochs | 6-8 | 8-12 |
| Batch size | 1024 | 4096 |
| Context number per user-game pair | 4 | 4 |
| Optimizer | Adam | Adam |
| Activation function | ReLU | ReLU |

1. Experimental results

* Evaluation metrics:
* ROC-AUC
* Precision - not used because data is imbalanced with around 85% negative instances in the US and 86% in Korea dataset
* Recall
* Results







* Training methods for the supervised component and unsupervised component: co-train and alternative train
* Co-train: simultaneously train the supervised loss function and unsupervised loss function
* Alternative train: alternately train the unsupervised component with the unsupervised loss function and the supervised component with the supervised loss function
* In general, co-train outperforms alternative train in terms of AUC under a different number of epochs (Fig 10) -> choose co-train as the final training methods

1. Related works (see paper)
2. Conclusion (see paper)