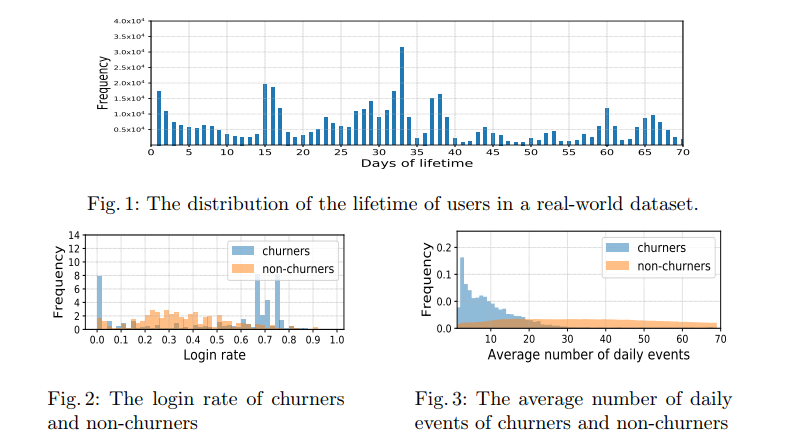
# ChurnPred

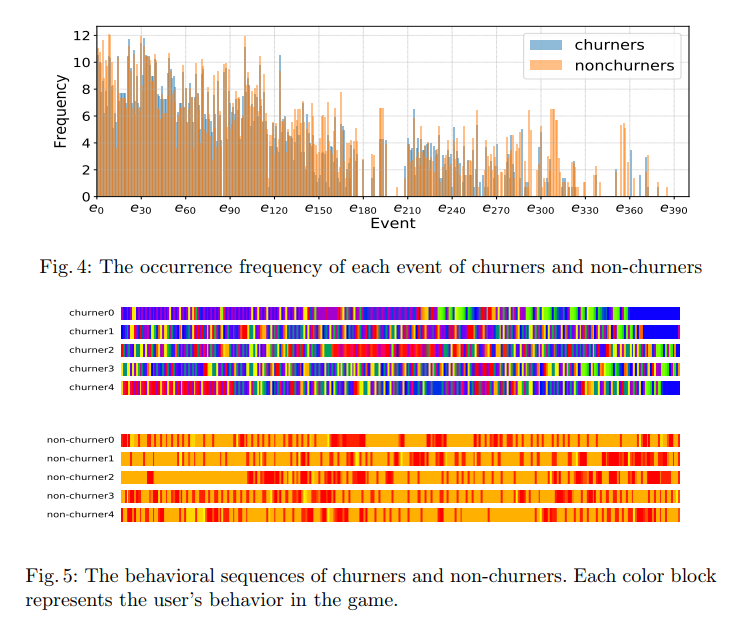
1. Introduction

* Several challenges of churn prediction in online games:
* Users’ behaviors – 2 aspects: login info and in-game behavioral info
* Different data types: login info – real-value vectors >< behavioral info – discrete value representing a specific action
* Challenging to model these data together to capture user-game interactions and inherent behavior patterns
* Each player has their own lifetime
* Short- and long-term modeling required to capture evolution of users’ preferences and temporal patterns
* Users’ behaviors – closely related to their daily life
  + E.g., weekdays vs. weekends
* Essential to additionally consider the influence of these info when modeling
* Novel end-to-end neural network approach: ChurnPred
* Consider login info and in-game info together
* Potential behavior patterns are automatically learned without manually extracting features
* Login info: impact of users’ lifetime -> LSTM models
* Time-aware filtering component to better distinguish characteristic behaviors based on period of events
* Multi-view mechanism to automatically extract multiple combinations of in-game behaviors from various perspectives which would lead to churn

1. Related works (see paper for more info)
2. Dataset description

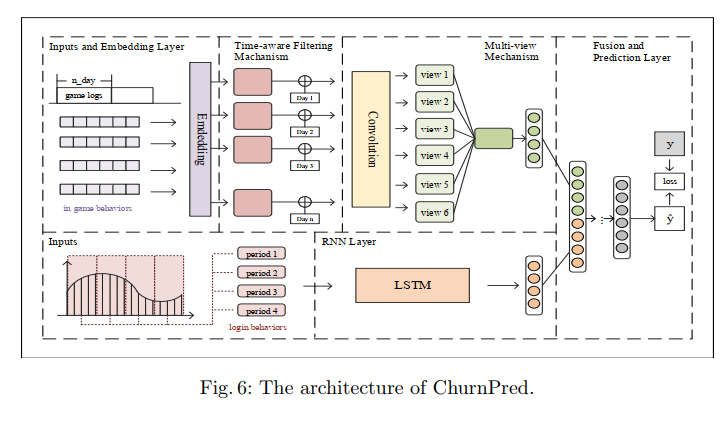
* Real-world dataset from a MMOPRG (NetEase Games)
* Collected from a server incl. user logs
* 485 regular events defined based on game content
* User logs automatically established for each player to record the events as well as the times when the player trigger them
* 2 classes of users defined: churners and non-churners
* Churners: consistently inactive for over 7 days
* : the day that user leaves the game
* Users whose leave\_day fall in churn window -> churners
* Users whose leave\_day are after observed\_day2 -> non-churners





1. Model architecture

* 3 main components:
* In-game behavior encoder: models in-game behavioral info of each user as a context embedding vector
* Login behavior encoder: models login info of each user as a context embedding vector
* Fusion and prediction layer: aggregates above 2 kinds of embedding vectors -> final possibility of whether user leaves the game



1. **In-game behavior encoder**

* **Inputs:**
* **Daily events** **for user u** and arrange them in chronological order -> **in-game behavioral sequences**

Where:

* User’s total historical behaviors are lengthy and massive -> greatly increase complexity and training time
* **Use data of as input**:
* **Embedding layer**:
* Given , events are embedded into **content vectors in a latent space** through an embedding layer
* **Each event identity** are **encoded into an one-hot vector** with |E| dimension
* Inputs are high-dim binary vectors -> **transform into dense representations**
* **Event embedding vector**: (1)

Where:

* **Time-aware filtering mechanism:**
* Time-aware gating mechanism to capture these characteristics behaviors based on time period
* **Gated linear unit (GLU)** for language modeling to allow the model selects related words or features for the next word
* Add some changes based on this structure
* Additionally consider the effect of period on the inputs
* **Time matrix for each day** -> **select what features will be propagated** to the downstream layers

(2)

Where

* **Multi-view mechanism:**
* Adopt **CNN units for multi-view generation**
* Regard as an **‘image’ of behavioral info** and the **sequential patterns as local features of this ‘image’**
* Summary the multiple combinations of behaviors in various views
* Use  **filters to encode the in-game behaviors of each day** respectively
* **Each filter** **slide over** as follows:

(3)

(4)

Where:

* Note: views from the above formulas do not contribute equally to final results
* **Attention mechanism** to address this problem – learn the attentive weights for multiple vectors
* Modify the formulas to learn the weights from multiple matrices (i.e., views)
* **Representation of final view** is formed by a **weighted sum of these generated views** calculated as follow:

(5)

(6)

(7)

Where

* To make the final view into a latent vector, use **max-pooling to summarize the characteristics of the final view**

(8)

Where:

1. **Login behavior encoder**

* **Inputs:**
* Login behaviors can be expressed as login frequency, play time, etc.
* We use **daily play time for each user** to describe the login info
* **Time window with size days** and consider  **consecutive time windows before the**
* **Input** can be expressed as a **sequence**

Where:

* **Recurrent neural network layer:**
* Apply a multi-layer LSTM for long- and short-term modeling
* **Each layer of LSTM computes as follows**:

(9)

(10)

(11)

(12)

(13)

(14)

Where:

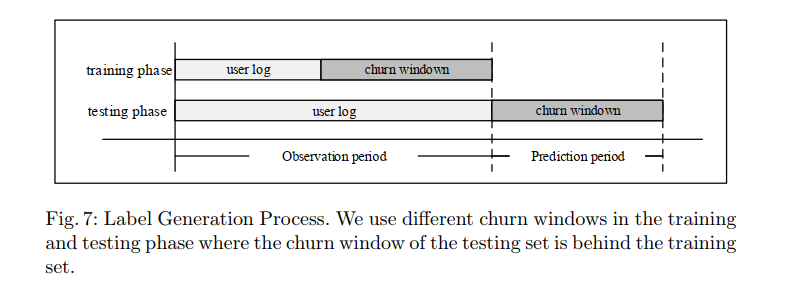
* Output of the last LSTM will be considered as the **context representation of login info**

1. **Fusion and prediction layer**

* After obtaining **2 kinds of context embedding**, i.e.
* **Concatenate these vectors into a unified vector** – high-level representation of behavioral features
* **Feed into a fully connected feed forward neural network**
* Output the **final prob for churn prediction**
* Unified embedding via fusion can be denoted as:

(15)

(16)



1. **Loss function and optimization**

* Adopt **cross-entropy** as loss function for model optimization
* To prevent overfitting, adopt **L2 regularization** on the params
* **Objective function:**

(17)

Where:

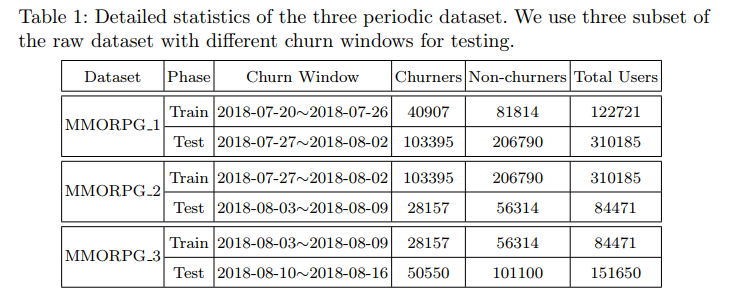
* Use **Adam optimizer** to learn the model

1. Experiment

* Research questions:
* RQ1: How does ChurnPred perform as compared with widely used methods and state-of-the-art ones in churn prediction?
* RQ2: What are the effects of the in-game behavior encoder and login behavior encoder in proposed method?
* RQ3: How do different hyperparam settings (e.g., dimension of embedding vectors) affect the performance of ChurnPred?

1. Dataset and experiment setup?

* Extract daily events of users in the game and arrange in chronological order serve as features of in-game behavioral info (behavior sequences)
* Length of each daily behavior seq. will be considered as the features of login info (i.e., daily playtime) preprocessed by normalization
* Constructing training and testing samples:
* Prevent data leakage
* Down-sampling approach applied to avoid skewed distribution (i.e., ratio of churners to non-churners is 1:2)
* Divide the dataset into 3 subsets where the churn windows in the testing phase are 3 consecutive weeks



1. Evaluation metrics and baselines

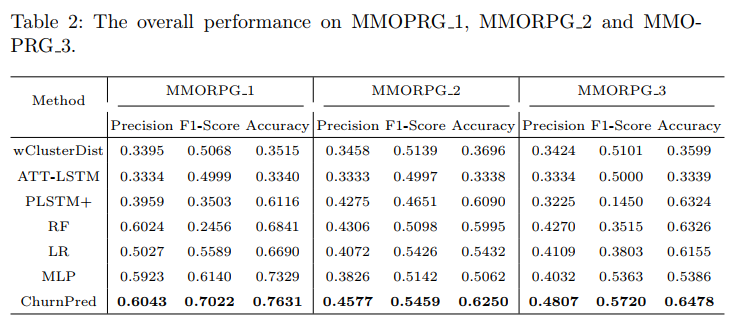
* Metrics: Precision, F1-score, Accuracy
* Each experiment is run 5 times to take the best F1-score as the final performance
* To verify the effectiveness of proposed ChurnPred model, compare with:
* Logistic regression (LR)
* Random Forest (RF)
* Multi-layer perceptron (MLP) – an artificial NN which maps a set of input vectors into a low-dim space. (implemented with 2 fully-connect layers with inputs of login info)
* wClusterDict: a distance-based classification schema conducted on login info as well as derived features in 3 semantic dimensions of engagement, enthusiasm, and persistence
* LSTM + Attention (ATT-LSTM): input is user behavior event seq. binned at constant intervals. Inputs are seq. of user in-game behaviors after registration
* PLSTM+: 2-step framework involving interpretable clustering and churn prediction. Prediction model based on LSTM by leveraging correlations among users’ multidimensional activities and the underlying user type is derived from the interpretable clustering.

1. Parameter settings

* Neural network-based models are all implemented in Pytorch, incl. ChurnPred, ATT-LSTM, PLSTM+, MLP
* Optimized by the Adam optimizer
* Batch size 512
* Hyperparams: Grid search for hyperparams on neural networks
  + Learning rate: {0.0001, 0.001, 0.01}
  + Size of hidden layer and embedding matrix: {8, 16, 32, 64}
  + Threshold: {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}
* For ChurnPred, convolution is done by convolution kernels with the width of 3 and height = size of embedding matrix
* For PLSTM+, set the lambda in loss function as 1 and use 2 hidden layers in each LSTM
* For wClusterDict, set the number of clusters at 5
* For ATT-LSTM, use 2-layer LSTMs and set the dropout as 0.5

1. Performance comparison (RQ1)

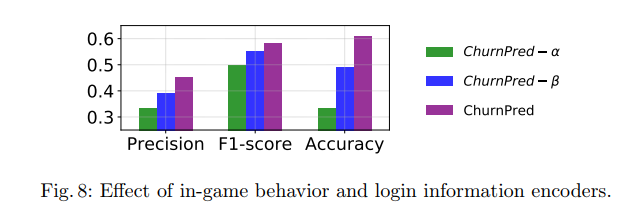
* Performance comparison of the proposed model and others



* Among conventional methods (LR, RF, and MLP):
  + RF – best average precision
  + MLP – best average F1 score
* RF is relatively correct on predicted churners >< fails to find out more true samples since it mistakes churners as non-churners in most cases
* MLP has higher F1 score – possible reason: model predict non-churners as churners as much as possible, which recalls more and more true samples and thus increases the F1-score with the decline in precision
* PLSTM+ and ATT-LSTM use in-game behavioral info of users as inputs >< different performance
  + PLSTM+ performs poorly -> frequency of users’ behavioral events is unable to fully describe recent behavioral info of users
  + ATT-LSTM uses behavioral seq. and achieves better performance -> potential behavior patterns in users’ behavior sequences have ability of indicating whether users leave the game
* RF, LR, MLP and wClusterDict all use login info as inputs
  + MLP perfoms the best on average F1-score metrics, followed by wClusterDict -> MLP can capture these dynamic changes in login seq.
  + RF and LR lack the ability to encode login seq.
  + wClusterDict benefits from derived features in 3 semantic dimensions which describes changes in users’ login status -> better performance
* ChurnPred: generally outperforms all baselines
  + Largely due to considerations on login info and in-game behavioral info in online games
  + Capture the dynamic characteristics in login info

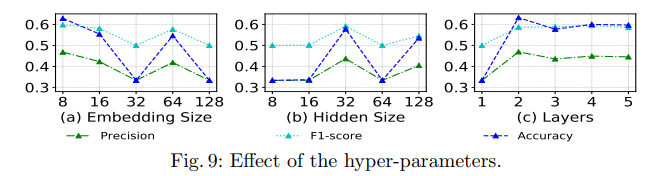
1. Component analysis (RQ2)

* Evaluate performance between login behavior encoder and in-game behavior encoder
* Design 3 different models:
* ChurnPred-α: retains login behavior encoder
* ChurnPred-β: uses only in-game behavior encoder
* ChurnPred: adopts above 2 kinds of components
* 3 models conducted on MMORPG\_1 dataset and keep the same model params when training
* In-game behavioral seq. implies potential behavior pattern, which contains more info about the user’s intention of leaving when compared with login info



1. Parameter sensitivity (RQ3)

* How different choices of params affect performance
* Except for the param being tested, set other params to default values
* Experiments conducted on MMORPG\_1 dataset



* Effect of embedding size:
* Model performance generally declines with increase of dimensions
* Proposed model is sensitive to dimension of embedding matrix
* Effect of hidden state dimension:
* Model have the best performance when hidden\_dim = 2 and perform poorly in most cases
* Dimension size of hidden state needs to be selected appropriately and otherwise it will get worse
* Effect of layer numbers:
* Best performance is obtained when n\_layer = 2
* After that, as number of layer increases, the perf begins to descent slowly and become stable
* 2 layers are enough for the model to achieve significant performance and more layers will not contribute to better performance

1. Conclusion and future work

* Future work: consider social influence on the in-game behaviors
* Scalability problem