Exam

Contents

Chapter 1

Introduction

1.1 Motivation

During the last few decades, we have witnessed the rise of web services offering any kind of goods, take for example Netflix for movies or Amazon for products. Such sites usually have huge catalogues of items that can overwhelm the user with too much information, making it hard for him to find items he would like; being able to narrow this large amount of contents is critical for these services, as it helps them generate greater incomes by suggesting users the right content to buy while also making them stand out from competitors as users find the service more useful.

1.2 Formalization

In the following, we will refer to any good a site can offer as *items* (e.g. movies, games, videos, news, etc); In a very general way, recommender systems are algorithms aimed at suggesting relevant items to users. Various approaches have been tried, among these we can define three families:

- *content-based* recommender systems, which create items and users profiles, embed them in a numerical feature space, and then suggest to the user the items which are nearest to him;
- *collaborative-filtering* recommender systems, which instead ignore content and rely only on user-item ratings; these furtherly divide in the way they suggest items to the user:
  - *user-based* approaches suggest him items liked by users similar to him;
  - *item-based* approaches instead suggest him items similar to items he liked;
- and finally *hybrid* recommender systems, which leverage both approaches.

Recently, recommender systems have taken in consideration sequential dynamics, seeking to capture patterns in the sequence of actions users perform. Contrarily to temporal recommendation systems which explicitly take into account the time of the actions, sequential ones only consider the order of actions, modelling sequential patterns which are independent of time.
As any sequence pattern recognition task, the problem is challenging, since the number of possible sequences grow exponentially with the number of past actions used as context. Markov-Chain

models overcome this issue by conditioning the next action only on the previous few actions, characterizing effectively short-range item transitions. To capture longer-range item dependencies, neural architectures have been used.
As we have seen, collaborative-filtering techniques rely on user-item ratings, which represent *explicit feedbacks* which are often not available. Other approaches therefore try to exploit *implicit feedbacks*, like clicks or purchases of the item.

Chapter 2

Data

In this project we will see how recommender systems can be leveraged to suggest *games*; to do this, we will use a large dataset of Steam reviews, which is publicly available online.

2.1 Description

The dataset contains information regarding both games and user-game reviews, separated in two tables *steam_reviews* and *steam_games*; here we can see the schemas
steam_reviews:

- **username**: reviewer username;
- **user_id**: reviewer id;
- **product_id**: reviewed game id;
- **text**: content of the review;
- **date**: date of the review;
- *found_funny*: number of users who found the review funny;
- *hours*: number of hours the user played the game;
- *page*: the page in which the review appears;
- *page_order*: ??
- *products*: ???
- *compensation*: ??

steam_games:

- *title*;
- *id*: game id;
- *developer*: company that developed the game;

- *genres*: genres of the game, *e.g.* action, adventure and so on;
- *metascore*: overall user score of the game;
- *price*: price of the game;
- *discount_price*: discounted price;
- *publisher*: company that published the game;
- *release_date*;
- *reviews_url*: url to the reviews of the game;
- *specs*: characteristics of the game;
- *tags*: tags of the game;
- *url*: link to the game;
- *app_name*: name of the application corresponding to the game;
- *sentiment*: overall sentiment of the game;
- *early_access*:

We will use two models, plus a naive baseline:

- *MF model*, which is a collaborative-filtering approach;
- *RNN-based*, which is a sequential recommendation approach;

Both don't require item profiles, so we will not exploit the rich game features in *steam_games*, which may be used in a content-based or hybrid approach.
Among the listed features in the review table, only the first 5 are needed for our approach, so after a bit of preprocessing in which we also replace the inconsistent ids with a unique progressive id, the situation is as follows

- **user_id**: reviewer id;
- **product_id**: reviewed game id;
- **text**: content of the review;
- **date**: date of the review;

so we have all we need to design the models.

2.2 Plots

The dataset exhibits some non-trivial properties, it is for example evident that the distribution of ratings follows a power-law both concerning users that games:

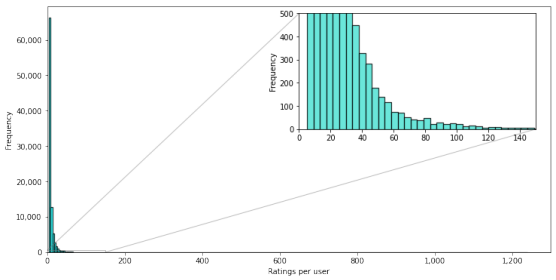

Figure 2.1: Ratings per user.

as we can see in the figure most of the users reviewed few games, while only few users have reviewed a significant number of games. The same thing happens for games, few games received a large amount of reviews, while most games have few.

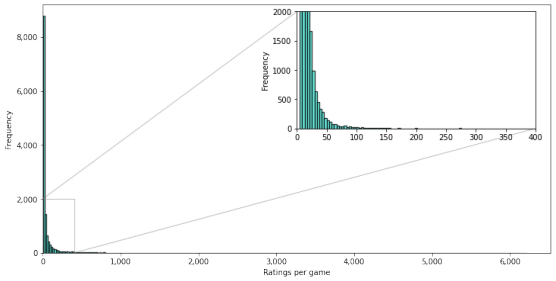

Figure 2.2: Ratings per game.

The distribution shows a long tail which amounts for a significant part of the catalogue, so a good recommender system should be able to recommend less famous games, even if it is in fact harder.

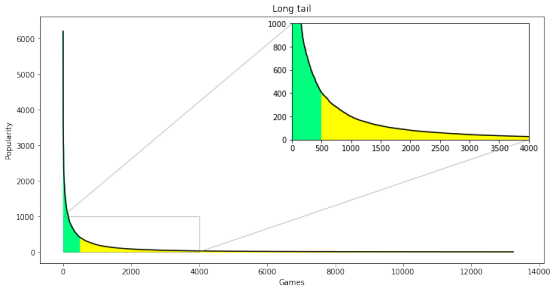

Figure 2.3: Long tail.

2.3 Preprocessing

Two of the models we will see require an explicit ratings matrix in input; nevertheless, our dataset only provides us text reviews, which must be numerically handled somehow. There are approaches which work directly on implicit feedbacks, but considering reviews as binary features which model whether the user-item interaction is present or absent doesn't make good use of the information, as a review is much richer and can be seen as a verbose rating.
For this task, we will use a pretrained *sentiment analyzer* from StanfordNLP, so every text review will be mapped to a value $r \in \{0, 1, 2\}$, encoding the sentiment as follows

- 0: negative;
- 1: neutral;
- 2: positive.

Allowing us to reduce the problem to the classical explicit feedback setting.
As we can see in the figure, reviews are fairly balanced, with most being neutral. Note that this may be true in the real distribution or a bias coming from the sentiment analyzer.

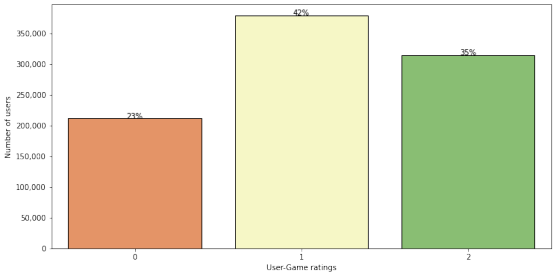

Figure 2.4: Sentiment distribution.

Chapter 3

Models

3.1 Popularity based

We will consider as a baseline a constant model, that is a model that suggests the same games to every user, independently from his tastes. As we don't use any information regarding the user, the safest bet is to just suggest the most popular games.

3.2 Matrix Factorization

The first real model that we will try employs a *Latent Factor CF* approach. In general, latent factor models are statistical models that relate a set of observable variables (so-called manifest variables) to a set of latent variables. In our case we want to predict user ratings by representing both items and users with a number of hidden factors inferred from observed ratings.
The basic assumption is that there exist an unknown low-dimensional representation of users and items where user-item affinity can be modeled accurately. *Matrix Factorization* is a way to obtain these lower-dimensional representations directly from the observed data.
In general, we want to infer the rating $r_{u,i}$ of user u to item i;
The framework works as follow:

- map both items and users to a *joint latent factor* d-dimensional space, so
  - each user will be represented by $\mathbf{x}_u \in \mathbb{R}^d$,
  - each item will be represented by $\mathbf{w}_i \in \mathbb{R}^d$;
- estimate $r_{u,i}$ by applying the dot product

$$\hat{r}_{u,i} = \mathbf{x}_u^T \cdot \mathbf{w}_i = \sum_{j=1}^{d} x_{u,j} w_{j,i} \tag{3.1}$$

- recommend to user u the k items for which the estimate is maximum.

Easier said than done, as we need a reasonable way to map users and items to these latent factor vectors.

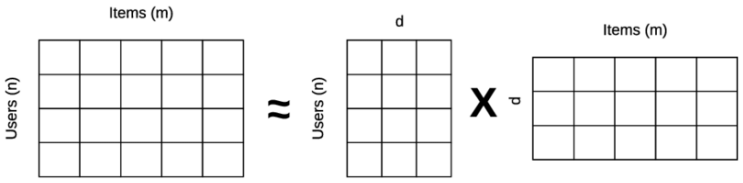

Figure 3.1: Matrix Factorization.

What we can do is leverage the observed ratings which are contained in the matrix R, and try to approximate R with the product of two matrices $X \in \mathbb{R}^{m \times d}$ and $W \in \mathbb{R}^{d \times n}$. This is equivalent to find the parameters that minimize the following loss function

$$\mathcal{L}(X, W) = \sum_{u,i \in D} \left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i\right)^2 \tag{3.2}$$

plus possibly a regularization term.
The resulting optimization problem can be solved either with SGD or ALS; The former is an iterative method which tries to minimize the loss function by descending its gradient, going opposite to the direction of steepest increase; nevertheless, this approach doesn't scale well when R grows large, therefore given the dimension of the ratings matrix in our scenario this may not be convenient.
ALS overcomes the non-convexity of the objective function by alternately fixing one latent vector and updating the other one; when one latent vector is fixed, the objective becomes quadratic and thus convex, allowing to find a closed-form solution.

3.3 Sequential

3.3.1 Formalization

To use sequential techniques we need to properly transform the dataset, obtaining for each user a vector

$$\mathbf{x} = (x_1, \ldots, x_n)$$

where x_i is the embedding of the i-th game reviewed by the user, ordered by timestamp. We want the model to be able to predict the next game that will be reviewed by the user given the previous reviewed games, that is predicting x_i given $x_1, \ldots, x_{i-1}$; To do this, we give the model the sequence of the first $n-1$ reviews $(x_1, \ldots, x_{n-1})$ as input and train it to predict the input shifted by 1 position $(x_2, \ldots, x_n)$. This is basically what in NLP is called *language modeling*, with the game ids composing the vocabulary.

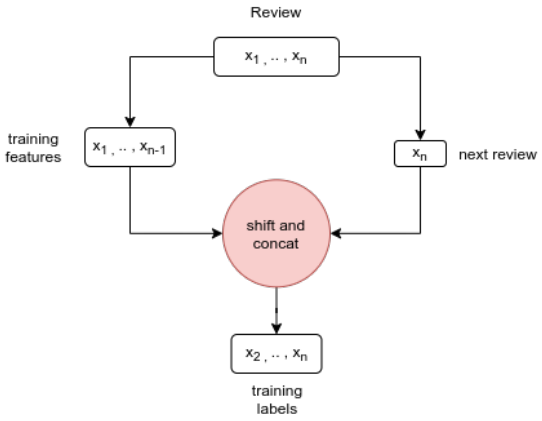

Figure 3.2: Input preprocessing.

Cross entropy is used as loss function, as each token prediction is a multilabel classification task

$$\mathcal{L} = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{3.3}$$

3.3.2 LSTM

As we have previously introduced, sequential recommender systems can exploit specialized neural architectures such as *Recurrent Neural Networks*; these summarize the context of a certain token m with a recurrently updated vector

$$\vec{h}_m = g(\vec{x}_m, \vec{h}_{m-1}), \quad m = 1, 2, \ldots, m$$

where $\vec{x}_m$ is the vector embedding of the token w_m and g defines the recurrence. Nevertheless, RNNs often fail to capture long-time dependencies; for this reason, LSTMs are often used. These employ a more complex recurrence, in which a memory cell goes through a series of gates, in fact avoiding repeated applications of non-linearity. The hidden state $\vec{h}_m$ accounts for information in the input leading up to position m, but it ignores the subsequent tokens, which may also be relevant to the tag y_m; this can be addressed by adding a second LSTM, in which the input is reversed. This architecture is called *Bidirectional LSTM*, and is one of the most effective neural architectures for sequences; nevertheless, for how we modeled the problem it would allow the model to cheat, as for any intermediate prediction it would be able to peek at the next game in the sequence and give it in output correctly.
Since the LSTM expects a batch of sequences of equal length, *padding* is added.

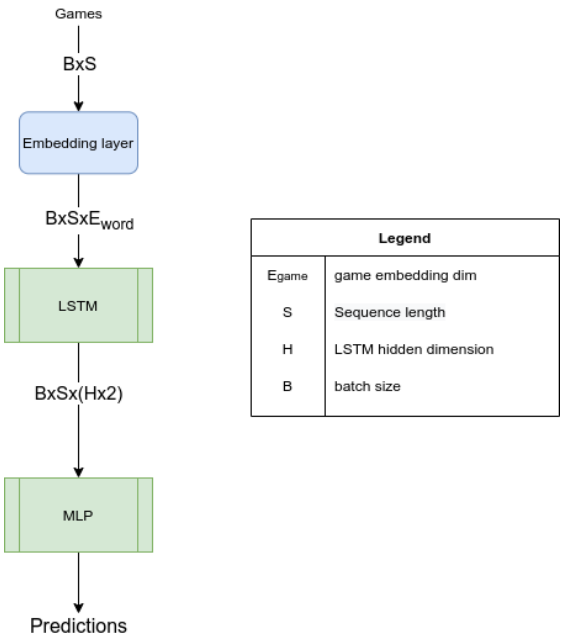

Figure 3.3: Model architecture.

The model thus works as follows:

1. each input sequence $\mathbf{x}$ is embedded by a word embedding layer;
2. it is then passed to a *LSTM* encoder which takes as input the embedded sequence and returns a dynamic representation of each game and its context;
3. the hidden representation is then given to a *Multi Layer Perceptron* that maps each game representation to the games space.

Chapter 4

Results

4.1 Metrics

Root Mean Squared Error has been used as metric to evaluate the Matrix Factorization model,

$$\text{RMSD} = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}}. \tag{4.1}$$

while *Hit@10* has been used for the sequential model, which is a *top-n* metric that counts the fraction of times that the ground-truth next item is among the top 10 recommended items.

$$\frac{\#\text{ hits}}{\#\text{ users}} \tag{4.2}$$

4.2 Results

4.2.1 Matrix Factorization

In the table we can see the *RMSE* for the MF model against the popularity based baseline; while the results may not seem impressive, it must be noted that the problem is in fact challenging: sentiment analysis is still an open research field, the pretrained model that we have seen is said to reach an accuracy of 70%, the resulting error pile-up with the error produced by the recommender system may be severe; moreover, the game reviews may be particularly difficult for the analyzer to get right, as gamers are often ironic and have their own niche vocabulary of words and meanings.

4.2.2 Sequential

In the table we can see the *hit@10* score for the sequential model against the popularity baseline; an accuracy of 0.503 would be low in a typical classification setting with few classes, especially if we consider that the model has 10 tries; nevertheless, in this scenario the model must be able to discriminate among 8590 classes; to understand what that means, let's see what score would

achieve a random baseline.

$$Pr\{hit\} = \sum_{i=1}^{10} Pr\{pred_i \text{ is correct}\} \tag{4.3}$$

$$= \sum_{i=1}^{10} \frac{1}{\#\text{ classes}} = 10 \cdot \frac{1}{8590} \approx 1e^{-4} \tag{4.4}$$

4.3 Study case

Chapter 5

Conclusions

We have seen two very different approaches to recommender systems, neither of them reached state of the art: for example, attention-based sequential recommender systems have reached an hit ratio at 10 of 0.7 over the same dataset. The MF approach we have seen is too naive and doesn't exploit all the available features which would make up for a good hybrid based recommender system, while the sequential approach using RNNs has few intrinsic flaws, for example the LSTM outputs a sequence of games where the same game is often repeated while it can't happen in the gold truth, nevertheless it is not easy to carve such constraints in the model. It would certainly be interesting to add content based features to both approaches, for example obtaining contextualized embeddings from the text reviews. For the sequential setting, it may be worth trying adding a sequence scorer on top of the LSTM, like a *Conditional Random Field*, to help assess the quality of a sequence of tags as a whole.