Movie Recommender System with LSTM

Yuqing Deng, Yujian Mi, Yujie Xu, Yiwei Sang

December 15, 2018

Abstract

People watch movies as a recreation in their spare time. However, it is difficult to select movies that suit our tastes, especially with a massive amount of movies made available by various movie platforms. In our project, we aim to establish a movie recommender system that make personalized movie recommendations to the users, by analyzing the movies that the users have previously watched in the past. We use Long Short-Term Memory (LSTM) to train, refine and experiment our model on Movielens dataset. The experiment results indicate high performance in terms of accuracy of predictions and generate reliable personalized movie recommendations.

1 Introduction

A recommender system is a type of information filtering system that attempts to predict the preferences of a user to an item, based on information that has been collected from the user, and then make suggestions based on these predictions. Recommender systems have become increasingly necessary in our daily lives. In current society, people are presented with an excessive amount of daily information, resulting in difficulty to effectively make decisions. Recommender systems appear as a solution to such information overload by suggesting users the most relevant products from a massive amount of data. Recommender systems have become quite ubiquitous in our lives. They are utilized in a variety of areas, ranging from selection of movies, music and books, to recommendations of friends and feeds on social media platforms, to professional research paper website. A recommender system typically makes recommendations through either content-based filtering or collaborative filtering. A content-based filtering makes predictions based on the description of the items and information on the preferences of users. A collaborative filtering makes predictions based on the past selections of the user and other similar users. The underlying assumption is that if two users have selected a same item in the past, they are likely to share common attitudes towards another item. In our project, we make predictions through collaborative filtering.

2 Main part

2.1 Datasets

In order to use RNN, we need a dataset that contains sequential information. We use the *MovieLens 1M* dataset that is available publicly from *grouplens.org* which contains time information. There are 3 files in the dataset - rating file, movie file and user file. The rating file contains 1,000,209 anonymous ratings by users and the timestamp which is the time when the user watched the movies, represented in seconds. The movie file contains the id, title and genre of about 3900 movies. The user file contains personal information of total 6040 users, including gender, age, occupation and zip-code. Since we only consider movieid, userid and time information, we only used the rating file. We loaded the file into a dataframe. The dataframe looks like Figure 1 below.

	UserID	MovieID	Rating	Timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291

Figure 1: rating data

2.2 Data Preprocess

We want a sequence data that has the format [user_id, first_movie_id, first_movie_rating, 2nd_movie_id, 2nd_movie_rating, ...]. To process the data, we first sorted the dataframe according to the timestamp column. Users and movies that appears in too few interactions are removed. If a user watched less than two movies, that user is removed from the dataset. If a movie is watched by less than five users, that movie is removed from the dataset. After the filtering, there are total 6040 users and 3706 movies remaining in the dataset. Then, we group rows by userID and generate sequences of user actions from data. Each sequence has the format [user_id, first_movie_id, first_movie_rating, 2nd_movie_id, 2nd_movie_rating, ...]. Now the dataset looks like Figure 2 below.

We shuffled the dataset and split it into training, validation and test with proportion of 80%, 10%, 10%.

0	0 2757 4 1444 4 1068 5 861 5 1978 3 1519 5 292...
1	1 998 4 1017 3 1010 4 2325 3 1091 5 2538 4 102...
2	2 549 3 2458 4 3039 3 1637 4 1630 4 1211 3 106...
3	5 2655 4 760 4 1675 4 1010 3 3039 4 881 4 16 4...
4	6 1068 4 446 4 408 4 996 5 1021 4 1666 5 3233 ...

Figure 2: data after preprocess

2.3 Model

Recurrent neural network (RNN) is a type of artificial neural network that deal with sequences of inputs. RNN is usually used as a language model, where each neuron is a word, and a sentence (a sequence of words) form a sequence of neurons in the order of the appearance of the word in the sentence. Long Short-term Memory (LSTM) network is a type of RNN that consists of LSTM units which is commonly composed of a neuron, an input gate, an output gate, and a forget gate. The three gates are used to control the flow of information into and out of the cell by remembering the values over arbitary time intervals. LSTMs are commonly used for time series data, where they generate predictions based on the the data of several past intervals. For example, LSTM is often used for natural language processing, where the input text is split into sentences(training examples) and each words in each sentence are tokenized to elements and feed into LSTM. In our project, we took a similar approach by considering all the movies watched by a single user, sorted by time stamp, as a sequence(training example) and each movie in the sequence as an element.

In our experiment, we first tried a unidirectional single-layered LSTM. In order to make the training example with same length, we used random target selection method with element size 30. For a sequence(all the movies watched by a single user) with length l, we randomly choose around $(\frac{1}{15})$ targets in the sequence. For each target, the previous 30 movies are used as a single training example. For example, we have a sequence m_1, m_2, ..., m_n. We randomly choose m_i as the target, so m_{i-30} to m_{i-1} are used as input. Thus, we have a single training example with X = m_{i-30} to m_{i-1} and Y = m_i. If the length is less than 30, we add padding zeros to the end. Then, we convert each element in the training examples to be one-hot encoded. That is, while we read in 30 movies by a user, we hope to predict the next movie watched by the user. Thus, we use many-to-one LSTM, that is we only care about the output of the LSTM unit.

We built the model by using Keras package. Keras is the deep learning library of Python. It is a high-level neural networks API that is capable on top of TensorFlow, which is an open-sourced library created by Google, that is available on Github. The first layer of our model is a unidirectional single-layered LSTM. The first parameter for LSTM is the

number of hidden units. Since we already preprocess the input and each traning example has the same length of input size 30. We set the number of hidden units to match the length of movies in one training sample. Then, in order to prevent overfitting, we set the dropout parameter to be 0.2, which is the fraction of the units to drop for the linear transformation of the inputs. We choose batch size 30 at first, that is we feed in 30 traning examples in each iteration. We then choose Adagrad [4] with a learning rate of 0.1. We calculate our loss using "categorical cross entropy".

The second layer of our model is dense layer, also called fully connected layer. In a dense layer, each node in the layer is connected to all the nodes in the preceding layer with corresponding weights. The dense layer represents a matrix vector multiplication. The values of the parameters in the matrix are trained during backpropagation. The Dense layer implements the operation: output = activation(dot(input, kernel) + bias), where activate is the element-wise activation function, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer.

Our final layer in the model is the softmax function. The softmax function normalizes the input vector into a probability distribution. Prior to applying softmax, the input vector is un-normalized - the elements could be greater than one, or be negative, and the summation of all the elements is not 1. After applying softmax, each element is the probability of each movie being a target. The value is between the interval [0,1], and $\sum_i x_i = 1$

We also tried a bidirectional single-layered LSTM, where we replaced the first layer of the model with a bidirectional LSTM. A bidirectional output layer can obtain information from both the past state and the future state. A bidirectional approach is especially useful when we need the context of the input.

3 Results

Metric Instead of using loss as a metric to evaluate our model, we took a more practice approach using the successful prediction rate due to the large amount of movies in our dateset. For each prediction, our model is given 30 movies as input X and 1 movie as target Y. We define a prediction as successful prediction if the target Y is in top 10 movies with highest probability predicted by our model.

Successful prediction rate We trained our model using 100,000 training examples. Our

Model	**learning rate**	**batch size**	**max length(# of unit)**	**spr**
Unidirectional	0.1	16	30	19.020%
Unidirectional	0.2	16	30	19.120%
Unidirectional	0.1	16	40	19.127%
Unidirectional	0.2	16	40	19.321%
Bidirectional	0.1	16	30	17.205%
Bidirectional	0.2	16	30	17.353%

successful prediction rate is measured using validation set. We tuned different hyper parameters and the successful prediction rate is shown in the table above. We plot the accuracy of the 0.1 learning rate per 100 epoch (Figure 3). The result shows that learning rate, and of hidden units has very little affect on the successful prediction rate. We stop at 100,000 training examples because the successful predication rate on validation set almost stops increasing.

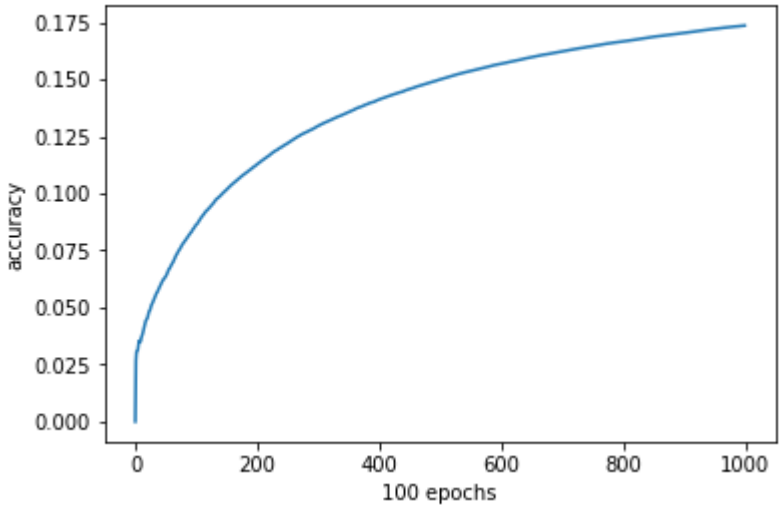

Figure 3: lr=0.1 for 100000 epochs

4 Conclusion

According to the result, the best model is unidirectional LSTM with learning rate 0.2, batch size 16 and number of hidden units 40, since it has the highest successful prediction rate. With 100,000 training examples, unidirectional LSTM does better than bidirectional LSTM in our case. One possible explanation is that the order of the movies are very important and previous movies help a lot in predict next movies while it is not true in the opposite way.

References

Robin Devooght, Hugues Bersini. Collaborative Filtering with Recurrent Neural Networks. arxiv, 2017.