UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Trip planning based on sequential recommender systems using textual representations

Author:
Bohdan YATSKIV

Supervisor:
PhD. Taras FIRMAN

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2022

Declaration of Authorship

"A year spent in Artificial Intelligence is enough to make one believe in God."

Alan Perlis

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Trip planning based on sequential recommender systems using textual representations

by Bohdan YATSKIV

Abstract

Finding interesting places to visit is one of the most common problems while travelling. With the development of review services, a large amount of personalized information is produced by the users. We believe that the information that review texts contain could be used to improve the personalized recommendations. This work focuses on creating a sequential recommender system that provides recommendations based on the chronological history of users' text reviews. The proposed model uses BERT text embeddings to get review text representations and a Transformer encoder to learn the context between items in sequence with the multi-attention mechanism. This approach allows to provide relevant recommendations based on the user's sequential behavior and makes the trip planning more comfortable.

Acknowledgements

First, I am extremely grateful to my supervisor Taras Firman for the main idea of the work and guidance throughout this project. Also, I would like to thank all the people at UCU who made my four years here a great experience. Finally, I want to express a huge thank to my family for supporting me throughout all the years of studying.

Contents

List of Figures

List of Tables

List of Abbreviations

ML	**M**achine **L**earning
NN	**N**eural **N**etworks
SRS	**S**equential **R**ecommender **S**ystems
MF	**M**atrix **F**actorization
NLP	**N**atural **L**anguage **P**rocessing
MAE	**M**ean **A**bsolute **E**rror
MSE	**M**ean **S**quared **E**rror
RMSE	**R**oot **M**ean **S**quared **E**rror
NDCG	**N**ormalized **D**iscounted **C**umulative **G**ain

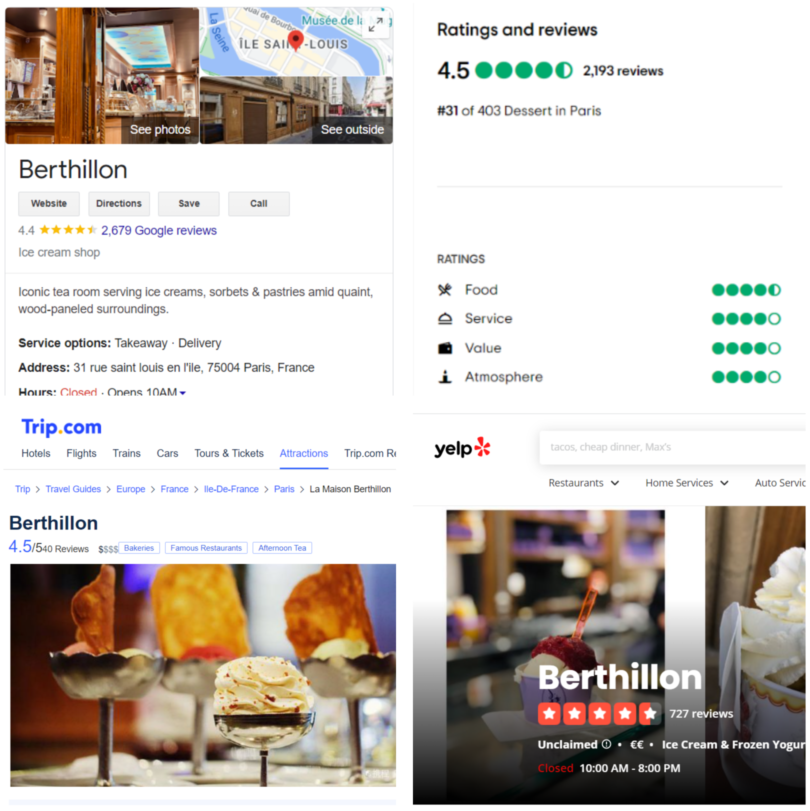
Dedicated to all Ukraine's defenders fighting for the freedom of our country

Chapter 1

Introduction

The Covid-19 pandemic has changed our lives in many ways. In 2020 massive tourism stopped almost completely. However, now when borders are open again, and most countries are loosening covid restrictions, travel-related problems become relevant again.

If you google the list of the most common questions during traveling, the route planning, and interesting spots to visit exploration will definitely be there. When you visit a new city, all the guidebooks usually propose quite the same popular routes and common places which are not based on your preferences. There are also plenty of services that help in seeking places, the most popular are Google, Tripadvisor, Yelp, Trip.com. They all provide reviews and "5 stars" ratings for businesses all around the world. For example, we can take a look at how the famous ice cream shop Berthillon in Paris, France looks like on all of these services.



Berthillon

4.4 ★★★★★ 2,679 Google reviews

Ice cream shop

Iconic tea room serving ice creams, sorbets & pastries amid quaint, wood-paneled surroundings.

Service options: Takeaway · Delivery

Address: 31 rue saint louis en l'ile, 75004 Paris, France

Hours: Closed · Opens 10AM

Ratings and reviews

4.5 ●●●●◐ 2,193 reviews

#31 of 403 Dessert in Paris

RATINGS

Food	●●●●◐
Service	●●●●○
Value	●●●●○
Atmosphere	●●●●○

Trip > Travel Guides > Europe > France > Ile-De-France > Paris > La Maison Berthillon

Berthillon

4.5/5 40 Reviews $$$$ Bakeries Famous Restaurants Afternoon Tea

Berthillon

★★★★★ 727 reviews

Unclaimed • €€ • Ice Cream & Frozen Yogur

Closed 10:00 AM - 8:00 PM

FIGURE 1.1: Example of same place page on different review services(Google, Tripadvisor, Trip.com, Yelp)

Although these review services can be helpful in getting information and general impressions about the place, they do not totally solve the problem of finding interesting places interesting for you. The 5-star ratings that review services provide are way too general and, as well as guidebooks, do not take into consideration your personal tastes and opinions. In addition, the ratings that users usually provide are quite polar. The most popular ratings among the users are 1 and 5. So this system of ratings is helpful in filtering places with principally negative reviews. But when the customer wants to compare some places with a reasonable number of positive reviews, the stars are not so representative in that case. Popular places will probably have around 4.5 stars, so it's often hard to compare such places based only on the rating. However, the 5-star rating is still the factor that matters the most to the consumers [Pitman, 2022]. Another interesting insight says that a bigger rating does not always mean better, and a perfect five score seems to be suspicious for most users [Collinger and Malthouse, 2015].

Hence there are plenty of problems that make the common rating system not perfect. Consumers will still need to check not only the rating but also read a couple of reviews and descriptions of the business and spend some amount of time discovering each place until they find one they like. All these problems, in addition with the idea of automatizing the process of getting place recommendations, are the reasons why we think that ML can be applicable and effective in this field. The task is to find some patterns in the review texts and star ratings. Here is where recommendation systems come in. Our goal would be to create a solution based on a sequential recommender system that learns from thousands of users' experiences and their text reviews and builds a unique sequence-route in each city, taking into consideration your previous history and preferences.

Chapter 2

Related works

2.1 Traditional recommender systems

By the definition, a recommender system is a set of algorithms that generates a personalized, optimized experience for a customer selected from discrete options [Burke, Felfernig, and Göker, 2011]. As the amount of information is increasing drastically and the problem of content filtering has become relevant as never. Recommender systems have become common in recent years and have a large range of usage on various websites such as streaming services or e-commerce services. The most common example is when a list of similar or related items is displayed under the current item you are exploring on some website. These lists are the result of recommender systems' work, and they try to follow the psychology that if a user chooses product A, he will choose product B.

There are two main paradigms for building classical recommendation systems: collaborative filtering and content-based filtering.

2.1.1 Content-based filtering

Content-based filtering focuses on similarities between items that users liked in the past. Usually, content-based systems take into consideration some metadata such as the description of the local business, the genre of the movie, the category of the product, e.g. These features are taken as input, and the output is user rating. Then this data is used to create a classification or regression model, which is specific to the current user. This model is used to predict whether the user will like some new items for which his ratings or buying behavior are unknown [Aggarwal, 2016].

This method has some advantages. Content-based filtering approach does not require other users' data for recommendations. It also works well for new or unpopular items as it does not require a lot of ratings for this item but finds similarities with other items the user has liked.

However, one of the main issues in the content-based approach is the quality of the features. The items that are going to be recommended the need to be described very well in order to receive some meaningful learning of user preferences. Moreover, it is supposed that every object is described at the same level, which is often not true as some items have missing features or are described in less detail than others [Burke, Felfernig, and Göker, 2011]. Another problem is that if the user is new and has no previous rating history, this approach is not able to work, so it is the reason why some apps often ask you to choose at least a couple of categories or themes you are interested in right after the registration.

Also, here comes the problem of overspecialization as if users do not have any ratings for items of some category, these categories will never be recommended even if the user can actually like them. So in this way, the user can always be kept in his own limited content profile and never be recommended anything outside of it.

2.1.2 Colaborative filtering

Instead of using items or user features, collaborative filtering analyzes relationships between users and their rating history to create some user-user, user-item, or item-item associations. The users' ratings data can be represented as a utility matrix with one axis being users and one axis being items.

User	Item1	Item2	Item3	Item4	Item5	Item6
A	2	3	3			5
B				5		
C		4	3			5
D	1	3	4	2		5
E			5		4	

TABLE 2.1: Example of utility matrix

There are also different methods inside the collaborative filtering approach.

Memory-based methods aim to find the closest neighbors, which means the most similar users or items. The difference from the model-based technique is that memory-based methods do not use any parametric machine learning algorithms and are based only on mathematical operations or transformations, including cosine similarity, Pearson correlation, or non-parametric algorithms such as KNN. Memory-based methods can also be divided into two categories: user-based and item-based.

- In user-based, similar users who have similar ratings for similar items are found for the target user. Then to predict the rating for an item unrated by the target user, a weighted average of the ratings of some k most similar users is computed. For example, in Table 2.1 for user A the system will define users C and D as the most similar as they rate Item3 and Item6 close to user A. Then, the prediction for user A rating for Item4 can be calculated based on users C and D ratings.

- In contrast, the item-based approach finds not the most similar users but the most similar items. Then a rating of an item can be predicted based on the ratings of a set of some k closest items. Different similarity functions such as adjusted cosine similarity or Pearson correlation are used to compare how similar the columns of the utility matrix are [Sarwar et al., 2001]. For example, in Table 2.1, Item2 and Item3 are the closest, so Item3 rating can be used to predict Item2 rating for user E.

The main problem of memory-based techniques is the bad scalability. When the amount of users is becoming big enough, the utility matrix is growing significantly, and the system requires loading a large amount of in-line memory. Then computational power becomes a bottleneck, and performance goes down [Do, Nguyen, and Nguyen, 2010].

Model-based methods solve this problem by using supervised or unsupervised machine learning algorithms to predict users' ratings. There are plenty of approaches such as decision trees, rule-based methods, Bayes classifiers, regression models, support vector machines, and neural networks. Almost any of the traditional machine learning methods can be generalized and optimized for the collaborative filtering problem as the classical machine learning regression and classification problems are actually just more specific cases of matrix completion problem, which is actually the same as what collaborative filtering is[Aggarwal, 2016].

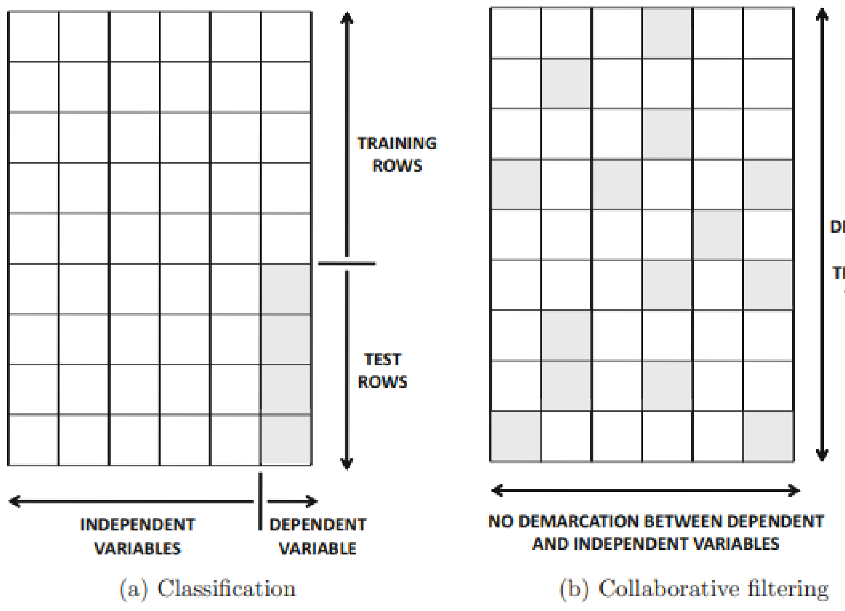

TRAINING ROWS
TEST ROWS
INDEPENDENT VARIABLES
DEPENDENT VARIABLE
(a) Classification
NO DEMARCATION BETWEEN TRAINING AND TEST ROWS
NO DEMARCATION BETWEEN DEPENDENT AND INDEPENDENT VARIABLES
(b) Collaborative filtering

FIGURE 2.1: Representation of classification problem and matrix completion problem. Source: [Aggarwal, 2016]

For example, we can present the classical classification problem as a matrix where the first columns are input features, and the label is the last column with some empty values. The difference is that data is not structured, and there is no difference between features and labels or training and test sets. So, collaborative filtering can be interpreted as a more general variant of well-known machine learning problems, which means classical methods can be adapted and also used to solve the matrix completion problem.

The most successful and popular realization of the model-based approach is using the matrix factorization (MF) method. The Basic MF method gives a characteristic for both items and users by vectors of factors derived from rating patterns. Then if the user and item have high correspondence between their factors, the item is recommended [Koren, Bell, and Volinsky, 2009].

It aims to discover some k latent features (factors). MF maps users and into same latent factor space $\mathbb{R}$ in which each item i is corresponding to some vector $q_i \in \mathbb{R}$ and each user u corresponding to vector $p_u \in \mathbb{R}$. Then for some user u vector p_u represents association between the user and latent features. Same for some item i vector q_i shows the item-factors association. Then interaction between user u and item i can be calculated as a dot product of corresponding vectors p_u and q_i:

$$\hat{r}_{ij} = q_i^T p_j = \sum_{k=1}^{k} q_{ik} p_{kj}$$

The task is to find the mapping of each user and item to corresponding vectors p and q. The most popular approach to achieve this is gradient descent which aims to minimize the following error - squared difference between real r_{ij} and predicted $\hat{r_{ij}}$

$$e^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{k} q_{ik} p_{kj})^2$$

2.2 Sequential recommender systems

With the development of technologies, recommender systems are also developing, becoming more complex, and using new approaches. Although collaborative filtering methods show themselves extremely effective in the recommendation, they work with users, items, and ratings as with a large static collection. Hence, they are good in the general preferences of users, but they do not take into account the order of user interactions. As a result, they are not able to figure out some recent preferences or short-time behavior. People's tastes and preferences in some areas are not always static and can change over time. Recent researches regarding recommender systems usually focus on working with sequential data[Xu, Liu, and Xu, 2019]. Systems that try to predict the next item based on a sequence of previous interactions are the most popular nowadays. You could see them on the majority of modern e-commerce, streaming services, and other websites. Basically, any website with a catalogue of some entities is the place where sequential recommendations can be used. They can be session-based[Wang et al., 2019a] meaning recommending some items based only on the current web session, next-basket[Li et al., 2021] meaning, for example, if you bought a phone, you would probably buy some phone accessories, or next-item[Hsu et al., 2016] that for example, recommend you next video or song.

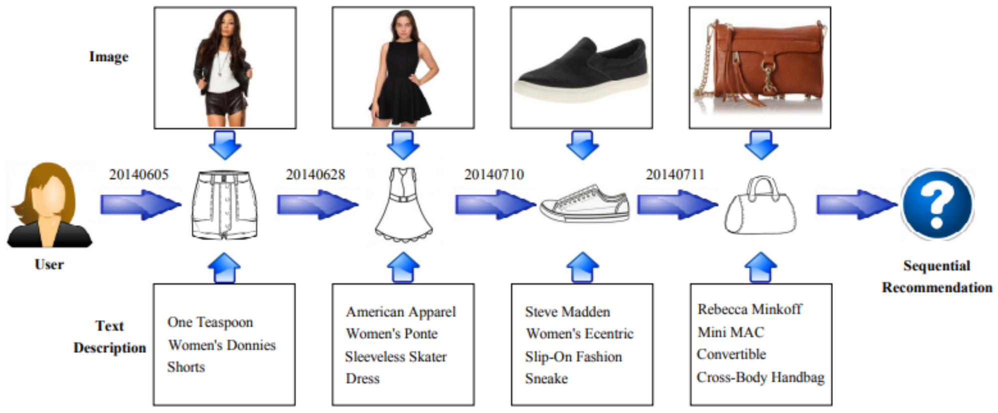


Image
User
20140605
20140628
20140710
20140711
Sequential Recommendation
Text Description
One Teaspoon Women's Donnies Shorts
American Apparel Women's Ponte Sleeveless Skater Dress
Steve Madden Women's Ecentric Slip-On Fashion Sneake
Rebecca Minkoff Mini MAC Convertible Cross-Body Handbag

FIGURE 2.2: Diagram of a user's purchase sequence. Source: [Cui et al., 2016]

The figure above shows an example of a sequential recommendation application. The goal is to predict an item the user would buy in the near future based on the sequence of items the user bought at a different time in the past. Text description, images, and other additional features associated with the items can be included to build a sequential recommendation model.

There are different approaches to the sequential recommendation. In general sequential recommender systems(SRS) can be divided into several categories.

- **Traditional sequence models** represent the simplest solutions, such as sequential pattern mining and Markov chain models. The implementation of these is relatively simple, but there is a number of drawbacks, such as finding some false patterns in case of pattern mining or disability to work with long-term sequences in case of Markov chain models.
- **Latent representation models** use latent features space. They develop the ideas of collaborative filtering MF and adapt them for the sequential recommendation.
- **Deep neural network models**. In the last years, neural networks(NN) have proven their strength in working with sequential data. Recent progress in natural language processing(NLP), computer vision, and image processing. The development of such NN models as RNN, LSTM, Attention models, and CNN make their usage in SRS the most advanced and popular choice.

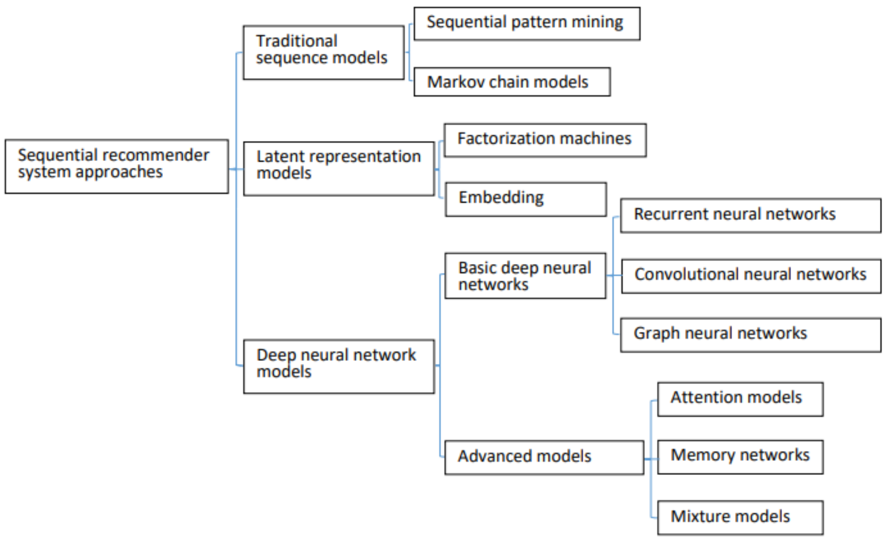

Sequential recommender system approaches
Traditional sequence models
Sequential pattern mining
Markov chain models
Latent representation models
Factorization machines
Embedding
Deep neural network models
Basic deep neural networks
Recurrent neural networks
Convolutional neural networks
Graph neural networks
Advanced models
Attention models
Memory networks
Mixture models

FIGURE 2.3: SRS classification. Source: [Wang et al., 2019b]

As Google paper[Vaswani et al., 2017] introducing the Transformer model(which will be described in the next section) made a huge impact in NLP and machine translation, attention-based models have become a new trend in the sequential recommendation. The main advantage of attention models is that they are able to give more importance to relevant interactions and less to irrelevant ones [Wang et al., 2019b].

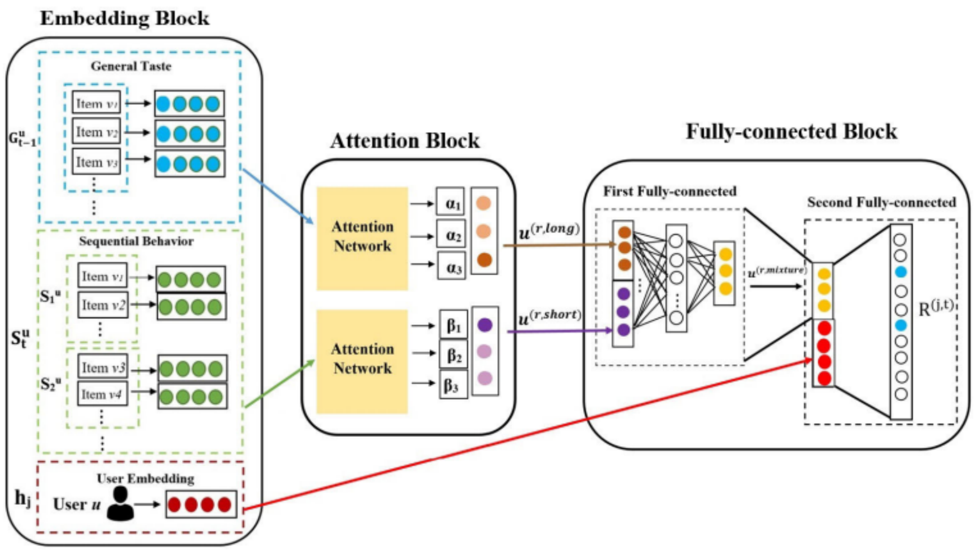


FIGURE 2.4: Example of attention based SRS architecture. Source: [Yakhchi et al., 2020]

The common SRS architecture using attention networks consists of an embedding block where the encodings of features and positional embeddings are made, an attention block that uses the attention mechanism and works with sequential data, and a fully-connected block to receive an output. On the Figure 2.4 an example of attention-based SRS architecture is provided. it combines both long-term preferences and short-term behavior sequences and contains two attention networks for each of these types of inputs, respectively. The user embedding is also included and passed directly to the fully-connected layer to make recommendations more personalized.

Chapter 3

Technical background

3.1 Transformer model

As it was said before, the Transformer model introduced by [Vaswani et al., 2017] has changed NLP a lot. The main problem of RNN and LSTM is that for sequences that are long enough, the context of an item that is far away from the current is lost. Also, the fact that in these types of models, the sequence is being processed item by item makes parallelization impossible, so the process of their training is quite slow.

3.1.1 Attention

The problem of losing context is solved by using the **attention** mechanism introduced by [Bahdanau, Cho, and Bengio, 2016]. Sequence to sequence models usually contain an encoder and decoder, which work with fixed-length vectors. In non-attention models, each sequence element is processed separately, and the decoder processes the final hidden state provided by the encoder. Here the problem with handling long sequences of data occurs because the longer distance between some relevant information and the current step is, the more probability that it would be rewritten by the next steps in the chain becomes. Basically, attention can be interpreted as an interface connecting the encoder and decoder, which uses a weighted sum of all of the hidden states provided by the encoder and passes it to the decoder. By doing this, the decoder can focus on relevant parts of the sequence.

The process of deciding how much attention should decoder pay to each sequence element and calculation of the weighted sum is done in the following way:

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j$$

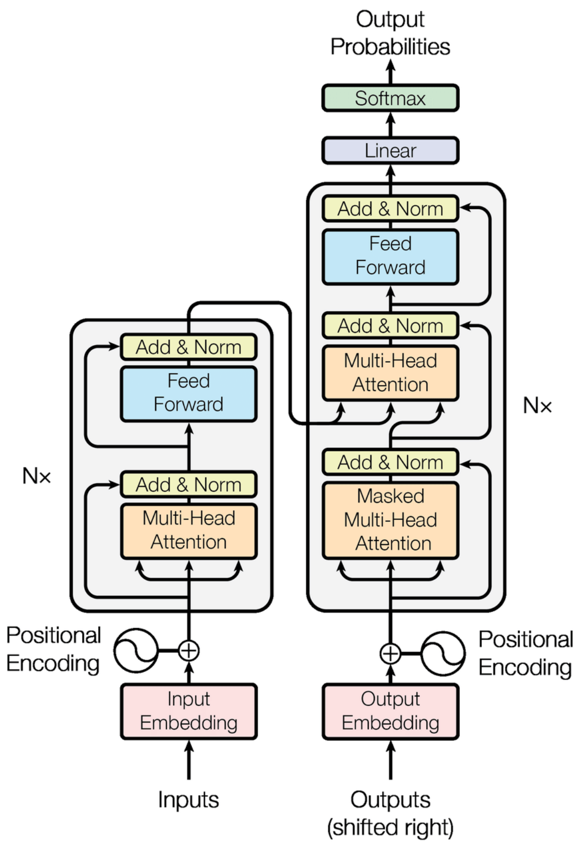
a context vector c_i is computed as a sum, where T is number of hidden states means length of sequence, h_j is jth hidden state, and $\alpha_i j$ is attention weight which is calculated by applying softmax operation to alignment score between input at j and output at i.

$$\alpha_{ij} = softmax(e_{ij})$$

e_{ij} is computed as an output of feed forward neural network denoted as function a where s_{i-1} is previous output of the decoder and h_j hidden state

$$e_{ij} = a(s_{i-1}, h_j)$$

3.1.2 Transformer model architecture



,

FIGURE 3.1: Transformer model

Attention solves the problem of losing context, but not parallelization. Transformer with its architecture solves it too.

The Transformer consists of stacks of 6 encoders and decoders. Each encoder consists of self-attention and feed-forward NN. Decoders consist of self-attention, encoder-decoder attention block, and feed-forward NN. The main point is that each item of the sequence is processed simultaneously, which makes parallelism possible. The self-attention mechanism works by computing attention for all items of the same sequence to encode the sequence. Basically, it helps to look at another sequence of items when encoding a single item to take context into consideration.

The inputs embedding received on first step is transformed into 3 vectors - queries, keys and values of dimension d. They are then stacked to matrices Q, K and V and attention is computed as:

$$Attention(Q, K, V) = sofrmax(\frac{QK^T}{\sqrt{d_k}})V$$

The Transformer also uses a Multi-head attention mechanism that linearly projects the queries, keys, and values to h dimensions using different learned projections. The attention function is performed in parallel, and then outputs of each attention block are concatenated to receive one result [Vaswani et al., 2017].

Self-attention blocks in the encoder take the Q, K, and V inputs as the outputs from the layer below. In the decoder, the self-attention layer is called masked multi-head attention because it allows using only earlier positions in the output sequence. It achieves it by setting future positions to $-inf$(masking them). The encoder-decoder attention is used to allow the decoder to focus on different parts of the encoder's outputs for each of its own outputs. It works the same as multi-head self-attention but takes Q from the previous decoder layer, while K and V ate taken from the encoders stack output.

3.2 Text encoding

As our system is using review texts as the input feature, the problem of representing texts as numerical values appears. In this section, we review the theory and existing approaches for natural language processing and word embeddings.

Word embeddings are vectors representation of the words. The length of the vector is the number of dimensions in which each word is encoded. Then these embeddings can be compared with cosine similarity to say how similar the words are. Word embedding is one of the key concepts of NLP.

By adding or subtracting the embeddings, interesting results can be achieved, so some logic can be built on top of this. The most famous example is "king"-"man"+"woman"="queen"

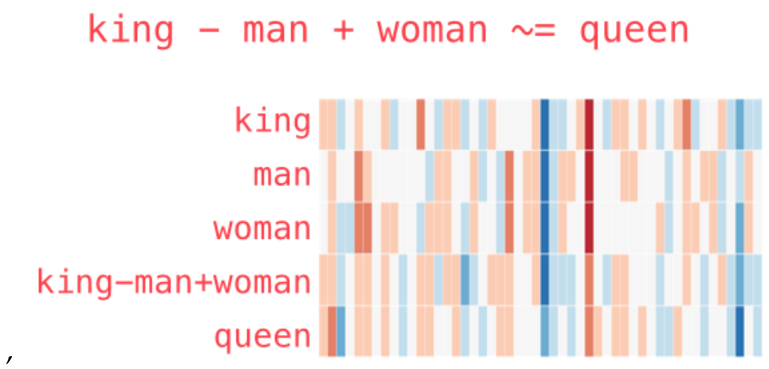


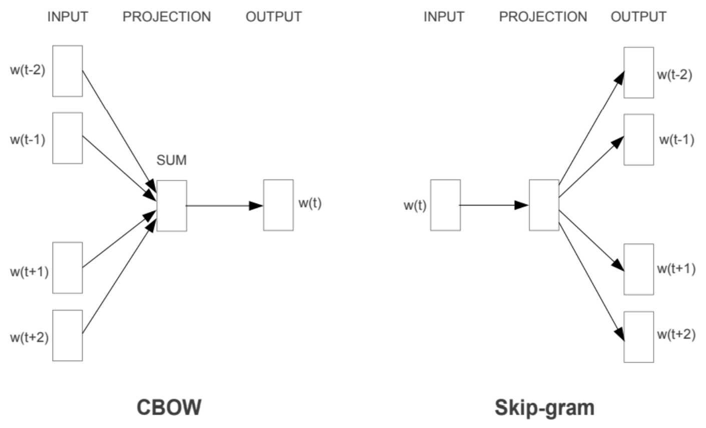
FIGURE 3.2: "king"-"man"+"woman"="queen". Source[Alammar, 2019]

In the Figure 3.2, we can see embedding vectors where numerical values are encoded by the color for visualization. The result of subtracting the embedding of the word "man" from the word "king" and adding "woman" after is very close to the word "queen" embedding.

3.2.1 word2vec

The word embeddings are created by looking at neighboring words in the text. The mechanic of language model training creates a sliding window of fixed size that slides along the text data from the training dataset. This sliding window generates a training sample. For example, if the size of the window is n, then the first $n-1$ words can be used as input features and the last one as an output, then this training set is used to train a NN.

Word2vec introduced by [Mikolov et al., 2013] proposes two new model architectures. First of all, word2vec takes into consideration not only previous words but also the following. The first model is called Continuous Bag of Words(CBOW) and adds not only n previous words but also n following to the train set inputs. The second proposed model works in an opposite way. It aims to predict the previous and following neighboring words based on the current word; it is called Skip-gram.



,

FIGURE 3.3: CBOW and Skip-gram models. Source[Mikolov et al., 2013]

Word2vec is also introducing a mechanism called negative sampling. The main idea is to reduce computation complexity by changing the problem from predicting the next word to the classification problem of whether two words are neighbors or not. It simplifies the task to a logistic regression which is a way less complex problem. However, in that case, if all the train examples are positive, the model can achieve 100% accuracy by always returning 1. To avoid this, a negative sample of not neighboring words is randomly chosen from the global words vocabulary and added to the train set [Alammar, 2019].

3.2.2 fastText

FastText introduced by [Bojanowski et al., 2017] develops the ideas of word2vec and takes into consideration subword information. It is based on the Skip-gram model, but each word is represented as a bag of character n-grams. It adds special tokens < and > meaning the beginning and the ending of the word. For example, for the word "queen" and n=3, the following sequences will be added to the vocabulary: <qu, que, uee, een, en> and the full-word sequence <queen> itself.

In such system of n-grams where **G** is set of n-grams for word w each word can be represented as sum of vector representation z_g of its n-grams,

$$s(w,c) = \sum_{g \in \mathbf{G}} z_g^T v_c$$

This helps the embeddings understand suffixes, prefixes, and, respectively, different forms of the same words and deal with out of vocabulary words.

3.2.3 BERT

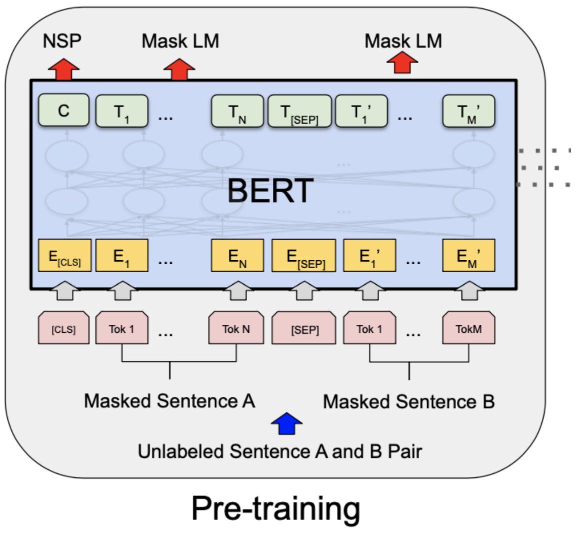
The introducing of Bidirectional Encoder Representations from Transformers(BERT) by [Devlin et al., 2019] was named as the beginning of a new era in NLP. BERT model showed itself extremely powerful in NLP tasks, broke several records, and still remains the leading model in terms of text representation.

BERT's key point consists in the application of the Transformer model to language modelling. Previous approaches were handling text sequences either from left to right or trying to combine left-to-right and right-to-left. The bidirectional approach allows making the context significantly deeper.

BERT uses Transformer to learn how words or sub-words contextually relate. As it was described in the previous section, the base Transformer model consists of encoder and decoder blocks, but the task of BERT is only to encode text to some representation, so only the encoder block of the Transformer is used. As the Transformer processes all the sequences at the same time and not word by word, so does BERT, simplified it is just a stack of Transformer encoders.

BERT uses two training strategies:

- **Masked Language Model(MLM)** takes a sentence with random words replaced with a [MASK] token. The goal is to fill that blanks and output the masked words. This helps BERT to understand the context between words in the sentence.
- **Next Sentence Prediction(NSP)** is a kind of binary classification problem. Two sentences are passed as the inputs, and the model predicts if sentence B follows sentence A. This mechanism helps BERT to understand the context between sentences which, together with MLM, means it can understand the language well.



,

FIGURE 3.4: BERT model architecture. Source[Devlin et al., 2019]

BERT is doing both these strategies simultaneously processsing two sentences. The embedding layer described in Figure 3.5 consists of the positional, sentence, and token embeddings. Positional embedding encodes the position of a word in the sentence, sentence embedding adds sentence A or B(as there are two sentences processed simultaneously), and token embeddings are the vocabulary ids for each of the tokens.

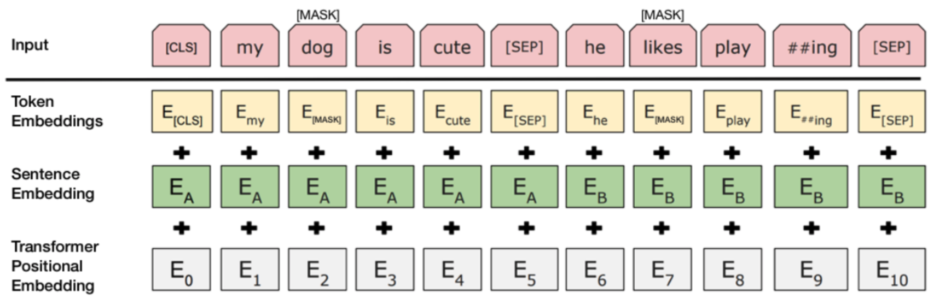

Input
[MASK]
[MASK]
[CLS] my dog is cute [SEP] he likes play ##ing [SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} $E_{[MASK]}$ E_{is} E_{cute} $E_{[SEP]}$ E_{he} $E_{[MASK]}$ E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Sentence Embedding
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Transformer Positional Embedding
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

FIGURE 3.5: BERT embeddings. Source[Devlin et al., 2019]

Then received embeddings are processed by the Transformer layer, and after classification and normalization layers, the output consists of C, which determines whether sentence B is following sentence A, and the same as in input number of T_{N+M} words including predicted masked ones.

Chapter 4

Proposed method

4.1 Dataset

In our solution, we use Yelp open dataset which contains around 7 million reviews from 2 million users for 150k businesses in 11 USA metropolitan areas.

4.1.1 Data preprocessing

The dataset consists of a file containing reviews of businesses and a file with the information about businesses themselves. The first step was to filter reviews only for places from one city, as our trip planner should give recommendations inside the chosen city. Philadelphia was chosen as the city with the most data. But the solution can be easily adapted for another city or dataset.

The dataset contained reviews not only in English but also some number of reviews in other languages. In order not to use multilingual text processing models, we decided to filter reviews only in English. To achieve this, pre-trained language identification model by fastText presented in [Joulin et al., 2016a] and [Joulin et al., 2016b] was used. Some outliers were also filtered, and category features were extracted from the places' information files.

After that process of creating the user behavior sequences was done. The sequence length was chosen as 8. The reviews were grouped by user and sorted by the date. For users that have more than eight reviews, several sequences were generated using a similar sliding window technique that is used in creating word embeddings. Users with less than reviews that chosen sequence length were filtered out.

The final sequences dataset for training is presented in a following way:

user id	place ids	review texts	ratings
0	[4450, 9672, 1981...	["Frequented this spot quite...	[3.0, 4.0, 2.0, 4.0...
1	[5773, 14427, 2293...	["I love Village Whiskey...	[4.0, 3.0, 5.0, 5.0...
...	...	...	...

TABLE 4.1: Sequences dataset sample

4.2 Model

The SRS architecture of our solution was inspired by the BST model presented by [Chen et al., 2019].

The model can be divided into three blocks. The embedding block encodes user ID and items sequence features into embedding vectors. Then the behavior sequence embeddings are passed to the Transformer block to learning some relations and contexts between the items in the sequence. Finally, outputs received from the Transformer layer are concatenated with user ID embedding, and the resulting vector is passed as the input to the fully-connected block, which consists of 3 linear layers and uses MSE loss to solve the regression problem and outputs the predicted user rating for the next item in the sequence.

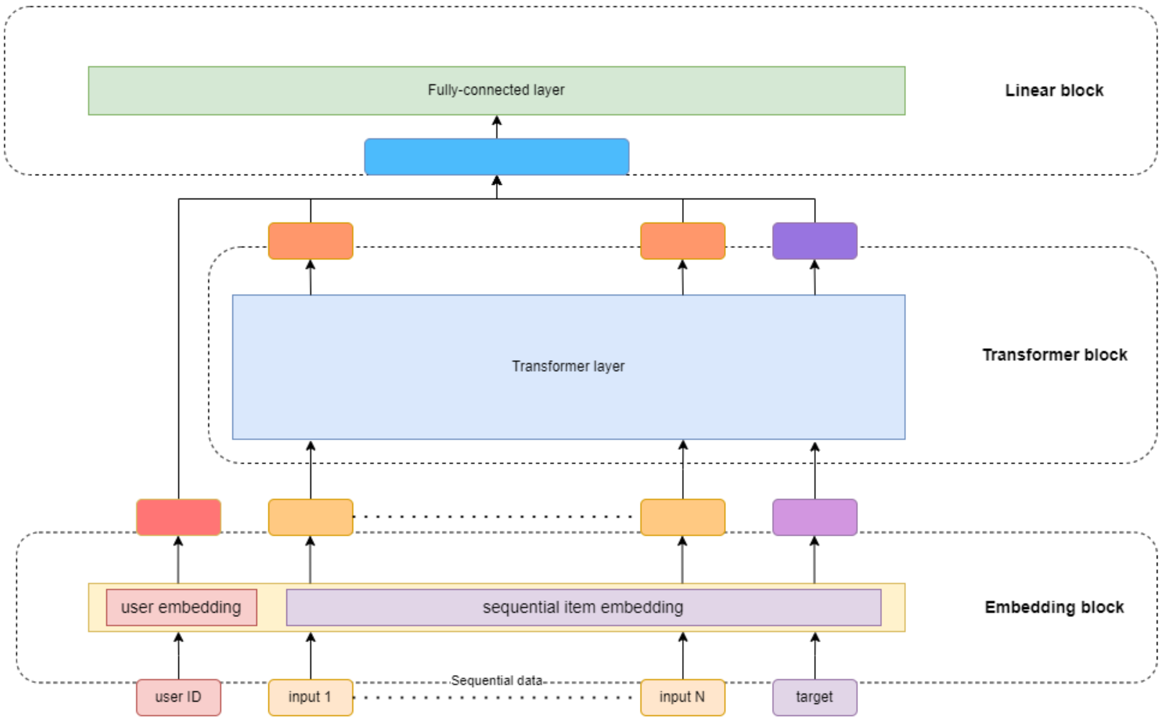


FIGURE 4.1: Model architecture

4.2.1 Embedding layer

The embedding layer is the most important part in our case, as it defines the fixed-size vector representation of sequential data, and actually, the feature engineering process happens here.

From the user features user ID is taken and embedding of vector size of square root of max user ID + 1 is created:

$$\dim(h_{user}) = \sqrt{max(userID)} + 1 \tag{4.1}$$

The items in the sequence are embedded in a more complex way. The review texts are encoded using pre-trained BERT language model presented by [Wang et al., 2020]. The SentenceTransformers Python framework was used as an engine.

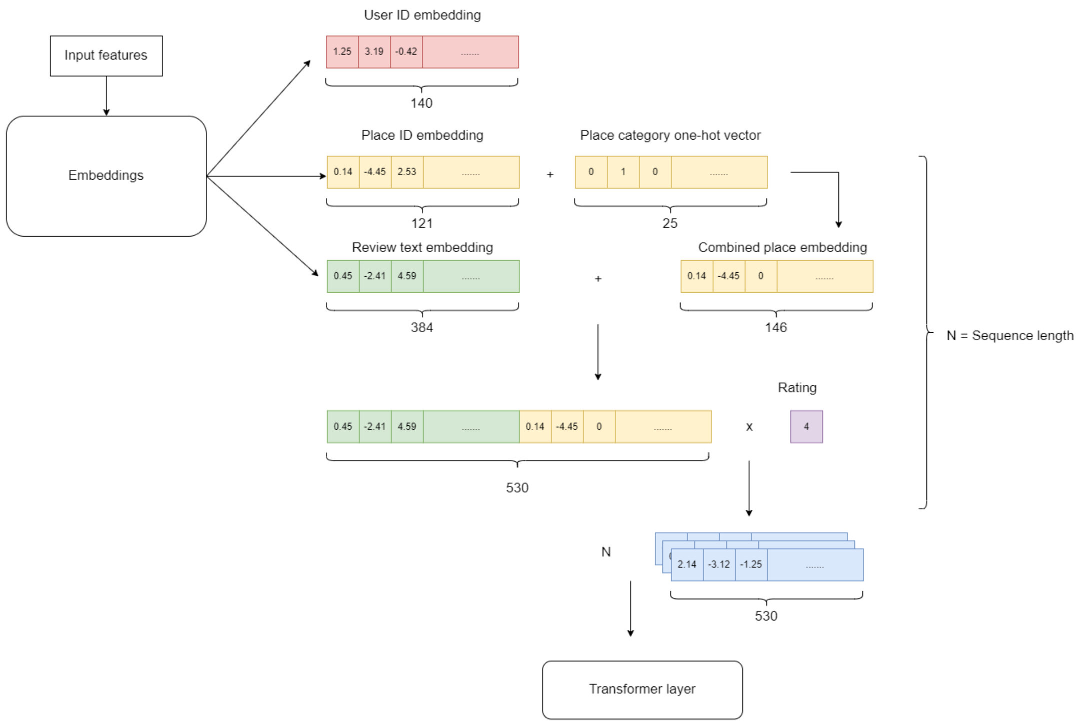


FIGURE 4.2: Embedding block

User and place IDs are transformed into vectors of lengths 140 and 121, respectively. Place ID embedding is concatenated with category one-hot encoded vector. Then embedded review text is concatenated with the resulting place features vector, and the received result is multiplied by the rating. These procedures are performed on N-dimensional matrices where N is sequence length(in the basic case is equal to 8), so doing this, we receive N vectors of length 530 which are then passed to the Transformer layer.

4.2.2 Transformer block

The Transformer block consists of the Transformer encoder that encodes the inputs using multi-attention mechanism. The parameter n-heads is by default set to 5 defining the number of multi-attention heads.

The N-1 input vectors and the target vector are processed by the transformer (where N is the sequence length). Then results are concatenated and adding the user ID embedding vector passed through the linear layer.

4.2.3 Linear block

The linear block consists of a fully connected network that takes the Transformer outputs and predicts the rating for the next item in sequence. The network itself consists of 3 hidden layers. Each of the layers consists of the Dense layer, Batch Normalization layer, Leaky ReLU activation layer and Dropout layer of parameter d (by default is equal to 0.2). The advantage of Leaky ReLU is fixes "dying ReLU" problem which is common in case of basic ReLU function. The loss function is mean squared error(MSE)

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y - \hat{y}_i)^2$$

The linear block schema looks in following way:

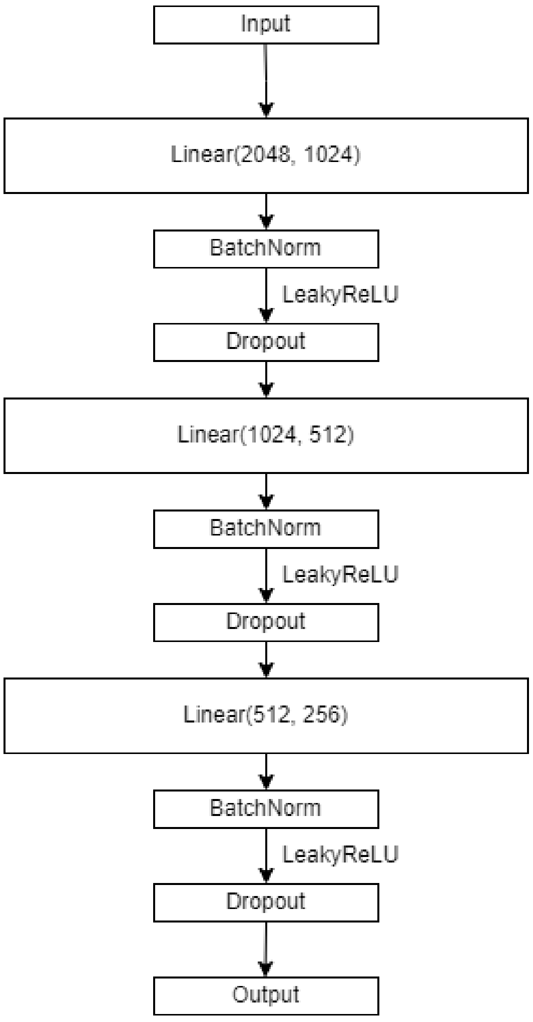


FIGURE 4.3: Linear block

Chapter 5

Experiments and results

5.1 Experimental setup

We used Python 3.7 and **PyTorch Lightning** library[Falcon, 2019], which is a high-level interface for popular PyTorch[Paszke et al., 2019] deep learning framework, in our solution.

In addition, following libraries were used:

- **fastText** [Bojanowski et al., 2016] - for language detection
- **sentence-transformers** [Reimers and Gurevych, 2019] - for creating the embeddings from the review texts

with some common Python libraries:

- **pandas** - for data processing
- **numpy** - for data processing
- **tensorboard** - for logging and metric monitoring

The model was trained using CUDA on NVIDIA GeForce GTX 1050Ti videocard.

Also, it was discovered that processing the text during the training process is not good in terms of performance because the same review texts can appear in multiple sequences. As a result, they are encoded multiple times, which makes the performance of the model poor. To avoid this, the text processing was done before the training, and all 887 159 review texts from the dataset were encoded in advance, and the resulting dataframe was saved in a .pkl file. The training data was then changed, and instead of review texts, as in table 4.1 the lists contain review IDs. Then the encoded review text can be retrieved by ID from the loaded processed texts dataframe.

5.2 Evaluation metrics

The process of evaluation of recommendation algorithms is complicated. In comparison with some other ML tasks, lack of some effective benchmarks for evaluation of recommendation systems is one of the problems the community is facing. Usually, an empirical study is conducted, and the model is compared to the state-of-the-art and several datasets and metrics are used.

As our work is aimed to trip recommendations and we use only the Yelp dataset, we compare our model with some other recommender systems evaluated on the Yelp dataset.

As the metrics for evaluation of regression and rating prediction we used MAE and RMSE.

However, goal of the recommender systems themselves is actually not to predict the rating but to recommend some items to the user. So in evaluation we also calculated widely used performance ratings for top-N recommendations:

Recall@N

Recall at N is the proportion of relevant items found in the top-N recommendations

$$recall@N = \frac{\text{\# of relevant items in } N}{\text{total \# of relevant items}}$$

Precision@N

Precision at N is the proportion of recommended items in the top-N recommendations that are relevant

$$precision@N = \frac{\text{\# of relevant items in } N}{N}$$

NDCG@N

We consider the item rating as relevance score. Then Cumulative Gain(CG) is the sum of all the relevance scores in N recommended items. To take into consideration the order of recommendations Discounted Cumulative Gain(DCG) is calculated by dividing relevance with the log of the position in recommended sequence.

$$DCG = \sum_{i=1}^{N} \frac{rating_i}{log_2(i+1)}$$

Normalized Discounted Cumulative Gain(NDCG) is computed as the ratio of DCG of recommendation to DCG of ideal possible order(iDCG).

$$NDCG = \frac{DCG}{iDCG}$$

To evaluate our model using these metrics for each user we take all existing sequences from the test dataset, and then predict the ratings for all items user has ratings for. We define rating 3.5 as a threshhold of relevant/irrelevant recommendation. Then the above metrics are calculated and then average is calclulated among all users.

5.3 Experiments

We run the training with different sequence length - 4 and 8. After some model parameters experiments we got the best result using sequence length 8 and following model configuration:

num of hidden layers	3
linear block dropout	0.2
Transformer d-model	530
Transformer n-heads	5
Transformer dropout	0.2
batch size	64
train epoch num	50

TABLE 5.1: Model configuration

5.4 Results

5.4.1 Regression metrics

With the selected model configuration we achieved **MAE**=0.7632 and **RMSE**=0.9749

We compared our results with a couple of researches that present recommender systems and are evaluated on the Yelp dataset [Kaluza, 2016]. Also CNN based solution (DeepCoNN) that uses review texts was compared[Zheng, Noroozi, and Yu, 2017].

	MF	SVD	SVD++	NMF	DeepCoNN	BiPartite	Our solution
RMSE	0.9807	1.0913	1.0952	1.1700	1.441	1.0863	**0.9749**
MAE	-	0.8568	0.8583	0.9033	-	**0.6663**	0.7632

TABLE 5.2: MAE and RMSE results

As we can see, our solution shows itself as the best RMSE result and second best in terms of MAE, where Weighted BiPartite Graph Projection [Sawant, 2013] shows significantly better results than all the other solutions.

5.4.2 Top-n metrics

We got average **precision@10**=0.8079 and **precision@20**=0.8017, which means that, on average, around 80% of items we recommend are relevant for users.

Precision@1	0.8117
Precision@5	0.8110
Precision@10	0.8079
Precision@20	0.8017

Recall@1	0.0555
Recall@5	0.1461
Recall@10	0.1968
Recall@20	0.2549

TABLE 5.3: Precision@N and Recall@N results

From the recall metrics, we can see that we recommend almost 20% of all relevant items in the first ten items of our recommended set.

NDCG@5	0.4303
NDCG@10	0.4281
NDCG@20	0.4275

TABLE 5.4: NDCG@N results

NDCG showed very good performance. However, to compare it with other solutions some further studies should be done because of the specifics of our testing experiment.

Chapter 6

Conclusions

6.1 Summary

In this work, we presented a sequential recommendation system for trip planning using user reviews textual information. We took the common architecture of attention-based SRS as a base and adapted it for working with text by extending the embedding layer with review texts embedding. The model demonstrated good performance in comparison with other approaches to the recommendation. But there is still a place for improvements to be made in order to approach to state of the art.

6.2 Future work

Our goal in the future would be to hold more experiments and try to apply some improvements in each block of the model. As our computational resources were not of the highest level, we could not afford some experiments in this work. First, the bigger-size text embedding model can be used to get better representations with bigger word embeddings vectors. With a faster process of training, a bigger number of epochs can be chosen, and more experiments with model optimizers can be done, as it seems like there is still a place to improve. In addition, the task can be expanded beyond the problem of travel recommendations, and the model can be adapted for solving other tasks based on text reviews. Some more datasets can be used. As a result, a more detailed evaluation and comparison with related works can be made.

Bibliography

Aggarwal, Charu C. (2016). *Recommender Systems: The Textbook*. Springer International Publishing. ISBN: 978-3-319-29659-3. DOI: 10.1007/978-3-319-29659-3.

Alammar, Jay (2019). *The Illustrated Word2vec*. URL: https://jalammar.github.io/illustrated-word2vec/.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2016). *Neural Machine Translation by Jointly Learning to Align and Translate*. arXiv: 1409.0473 [cs.CL].

Bojanowski, Piotr et al. (2016). "Enriching Word Vectors with Subword Information". In: *arXiv preprint arXiv:1607.04606*.

— (2017). *Enriching Word Vectors with Subword Information*. arXiv: 1607.04606 [cs.CL].

Burke, Robin, Alexander Felfernig, and Mehmet H. Göker (2011). "Recommender Systems: An Overview". In: *AI Magazine* 32.3, pp. 13–18. DOI: https://doi.org/10.1609/aimag.v32i3.2361.

Chen, Qiwei et al. (2019). *Behavior Sequence Transformer for E-commerce Recommendation in Alibaba*. DOI: 10.48550/ARXIV.1905.06874. URL: https://arxiv.org/abs/1905.06874.

Collinger, Tom and Edward C Malthouse (2015). *From Reviews to Revenue Volume 1: How Star Ratings and Review Content Influence Purchase*. URL: https://www.powerreviews.com/wp-content/uploads/2019/02/From-Reviews-to-Revenue-Northwestern-Report-Volume-1.pdf.

Cui, Qiang et al. (2016). *MV-RNN: A Multi-View Recurrent Neural Network for Sequential Recommendation*. DOI: 10.48550/ARXIV.1611.06668. URL: https://arxiv.org/abs/1611.06668.

Devlin, Jacob et al. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv: 1810.04805 [cs.CL].

Do, Minh-Phung, Dung Nguyen, and Loc Nguyen (Aug. 2010). "Model-based approach for Collaborative Filtering". In.

Falcon, William (2019). *PyTorch Lightning: The lightweight PyTorch wrapper for high-performance AI research. Scale your models, not the boilerplate.* DOI: 10.5281/zenodo.3828935. URL: https://www.pytorchlightning.ai.

Hsu, Kai-Chun et al. (2016). *Neural Network Based Next-Song Recommendation*. DOI: 10.48550/ARXIV.1606.07722. URL: https://arxiv.org/abs/1606.07722.

Joulin, Armand et al. (2016a). "Bag of Tricks for Efficient Text Classification". In: *arXiv preprint arXiv:1607.01759*.

Joulin, Armand et al. (2016b). "FastText.zip: Compressing text classification models". In: *arXiv preprint arXiv:1612.03651*.

Kaluza, Clara De Paolis (2016). "Recommender System for Yelp Dataset CS 6220 Data Mining Northeastern University". In.

Koren, Yehuda, Robert Bell, and Chris Volinsky (2009). "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42.8, pp. 30–37. DOI: 10.1109/MC.2009.263.

Li, Ming et al. (2021). *A Next Basket Recommendation Reality Check*. DOI: 10.48550/ARXIV.2109.14233. URL: https://arxiv.org/abs/2109.14233.

Mikolov, Tomas et al. (2013). *Efficient Estimation of Word Representations in Vector Space*. DOI: 10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Pitman, Jamie (2022). *Local Consumer Review Survey 2022*. URL: https://www.brightlocal.com/research/local-consumer-review-survey/#.

Reimers, Nils and Iryna Gurevych (Nov. 2019). "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. URL: https://arxiv.org/abs/1908.10084.

Sarwar, Badrul et al. (2001). "Item-based collaborative filtering recommendation algorithms". In: *Proceedings of the 10th international conference on World Wide Web*, pp. 285–295.

Sawant, Sumedh (2013). "Collaborative Filtering using Weighted BiPartite Graph Projection". In.

Vaswani, Ashish et al. (2017). *Attention Is All You Need*. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

Wang, Shoujin et al. (2019a). *A Survey on Session-based Recommender Systems*. DOI: 10.48550/ARXIV.1902.04864. URL: https://arxiv.org/abs/1902.04864.

Wang, Shoujin et al. (July 2019b). "Sequential Recommender Systems: Challenges, Progress and Prospects". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, pp. 6332–6338. DOI: 10.24963/ijcai.2019/883. URL: https://doi.org/10.24963/ijcai.2019/883.

Wang, Wenhui et al. (2020). *MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers*. arXiv: 2002.10957 [cs.CL].

Xu, Mingming, Fangai Liu, and Weizhi Xu (2019). "A Survey on Sequential Recommendation". In: *2019 6th International Conference on Information Science and Control Engineering (ICISCE)*, pp. 106–111. DOI: 10.1109/ICISCE48695.2019.00031.

Yakhchi, Shahpar et al. (2020). "Towards a Deep Attention-Based Sequential Recommender System". In: *IEEE Access* 8, pp. 178073–178084. DOI: 10.1109/ACCESS.2020.3004656.

Zheng, Lei, Vahid Noroozi, and Philip S. Yu (2017). *Joint Deep Modeling of Users and Items Using Reviews for Recommendation*. DOI: 10.48550/ARXIV.1701.04783. URL: https://arxiv.org/abs/1701.04783.